Arduino da zero

Crea il tuo primo progetto in 7 giorni

Benjamin Spahic

Contenuto

1. Prefazione e introduzione 1
2. Fondamenti di ingegneria elettrica 6
2.1. Il potenziale elettrico φ 6
2.2. Il voltaggio U 6
2.3. La corrente I 7
2.4. L'energia E e la potenza P 7
2.5. La resistenza R 8
2.6. Il diodo 10
2.7. Il condensatore e la capacità C 12
2.8. Tecnologia digitale 13
3. La piattaforma Arduino spiegata 20
3.1. Cos'è Arduino 20
3.2. La famiglia Arduino in sintesi 22
3.3. Struttura di Arduino Uno 24
3.4. Altro hardware 30
3.5. Scudi Arduino 30
3.6. Sensori e attuatori 30
3.7. Breadboard e cavo (jumper) 31
4. Basi della programmazione 34
4.1. Commenti e istruzioni 35
4.2. Variabili 37
4.3. Funzioni 42
4.4. Cicli e condizioni 42
4.5. Librerie 46
5. L'IDE di Arduino 48

5.1. Struttura di un programma Arduino 49
5.1.3. Aggiungere librerie 50
5.2. Indent 51
5.3. Funzioni importanti 52
6. I primi progetti 58
6.1. Controllo dei LED - controllo dei semafori 58
6.2. Controllo del servomotore tramite potenziometro 69
6.3. Sensori di controllo - Stazione meteorologica 76
7. Gli errori più comuni dei principianti 96
7.1. Cortocircuiti, fumo, calore 96
7.2. Il controller si blocca - i componenti non funzionano 96
7.3. La comunicazione bus non funziona 97
7.4. Il led non si accende 97
7.5. Nessun file o directory di questo tipo 98
7.6. "Atteso ';' prima di ' ... ' gettone" 98
7.7. "Atteso inizializzatore prima di ' ... ' " 99
7.8. "Dichiarazione attesa prima di ' ... ' token 100
7.9. "Ridefinizione di ' ... ' " 100
7.10. "Problema di caricamento sulla scheda" 101
8. eBook gratuito 104

1. Prefazione e introduzione

Nel 1941, il mondo era in rovina, ma due ingegneri tedeschi presentarono un'invenzione che influenzò la nostra vita più significativamente di qualsiasi altra scoperta del tempo. Stiamo parlando dello "Zuse Z3", il primo computer digitale del mondo.

Naturalmente, lo Zuse Z3 non può essere paragonato ai computer di oggi, né nella sua costruzione né nella sua funzione o nella sua dimensione spaziale.

Lo Zuse Z3 pesa diverse tonnellate e occupa quasi un'intera stanza. Allo stesso tempo, la sua potenza di calcolo è molte volte inferiore a quella di una moderna calcolatrice tascabile. Tuttavia, è considerato una pietra miliare nella ricerca informatica.

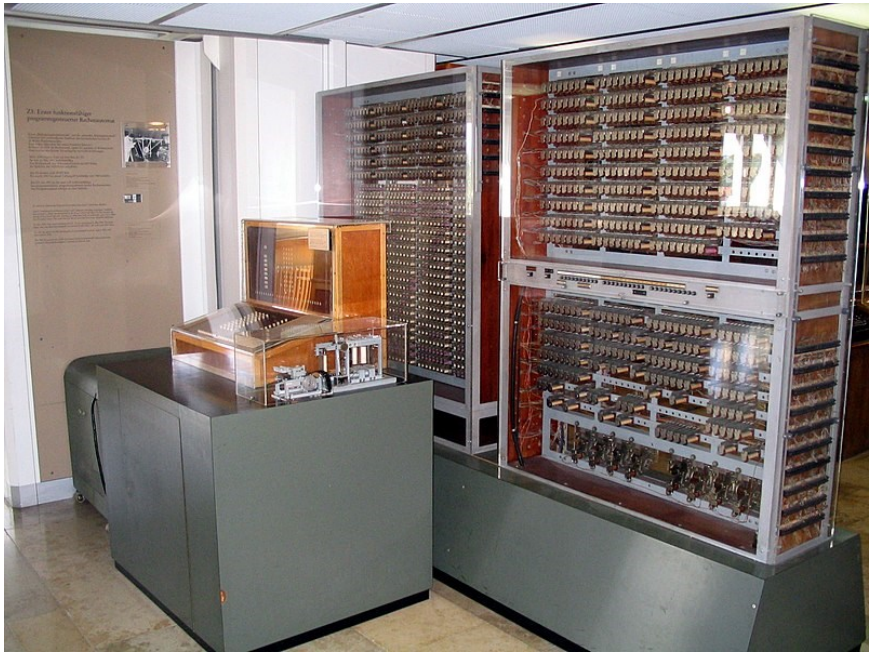


Illustrazione 1 Replica della Zuse Z3 nel Deutsches Museum di Monaco

Lo Zuse Z3 fu distrutto durante la seconda guerra mondiale e, quando il suo inventore Konrad Zuse fu arruolato per la guerra, lo lasciò pronunciando le seguenti parole:

"Altri lasciano la famiglia, io lascio Z3".

-Konrad Zuse

Molte rivoluzioni tecnologiche si sono susseguite dalla creazione di Zuse Z3. Nel 1946, l'esercito statunitense introdusse l'ENIAC, il primo computer completamente elettronico. Nello stesso periodo venne messo in funzione lo Zuse Z4, il successore dello Z3.

Ad oggi, la potenza di calcolo si è moltiplicata. Mentre lo Zuse Z3 aveva solo 600 unità di calcolo (relè), un processore moderno ha diversi miliardi di unità di calcolo (transistor) su una superficie di 200 mm^2, riducendo al contempo considerevolmente il consumo di energia.

Questo progresso tecnologico ci permette di condurre la vita che facciamo. Lavoriamo davanti agli schermi, scorriamo i social media sui nostri smartphone e la sera guardiamo comodamente serie TV o film.

La digitalizzazione è stata, e è ancora, la più grande rivoluzione della storia moderna. Ci sono infinite possibilità a nostra disposizione, ma c'è un serio problema di entusiasmo nella nostra società. Tutti usano la tecnologia e beneficiano del progresso, ma pochi hanno la volontà di capire l'ingegneria elettrica e la programmazione e di diventare parte del processo inventivo.

Tuttavia, dato che hai comprato questo libro, sembra che tu sia interessato proprio a questo argomento. Questo ci porta al tema centrale di questo libro: Arduino. La piattaforma Arduino è perfetta per realizzare i propri progetti, imparare ad utilizzare i microcontrollori e la programmazione e risvegliare lo spirito di scoperta.

Forse sei ancora uno studente che vuole programmare il suo robot, un programmatore esperto che vuole imparare di più sull'hardware, o un giovane pensionato che vuole imparare di più sui microcontrollori e la loro programmazione in generale.

In ogni caso, non ti pentirai di aver preso in mano la questione.

Ci sono certamente molti siti web e libri che ti introducono all'argomento Arduino. Tuttavia, alcuni di questi sono lunghi più di 500 pagine e sono

completamente inadatti ai nuovi arrivati.

Se hai già familiarità con la materia e hai bisogno di indicazioni per un progetto specifico, questi libri possono esserti utili, ma per una gran parte degli interessati non sono efficaci.

Ed è proprio questo problema che ha dato origine a questo libro.

È una guida per principianti e per persone curiose che vogliono capire i principi di base della piattaforma Arduino il più rapidamente possibile senza molte conoscenze precedenti, iniziando il loro primo progetto il più presto possibile.

Ma cos'è un microcontrollore? Come faccio a collegare correttamente i sensori e altri componenti al microcontrollore? Come posso programmare in modo rapido ed efficiente? Tutto questo viene elaborato passo dopo passo e completato con un progetto di esempio reale.

Prerequisiti e livello di conoscenza:

Questo libro è adatto a chiunque abbia un entusiasmo di base per la tecnologia e la programmazione. È impartante avere una conoscenza di base della fisica, come per esempio del calcolo con le unità SI, e quindi il concetto viene ripetuto.

Inoltre, utilissime sono le basi dell'ingegneria elettrica in quanto ti consentiranno di comprendere schemi di circuiti più complessi e flussi di corrente.

Poiché alcuni lettori potrebbero non avere alcuna esperienza con il linguaggio di programmazione "C/C#" o con i circuiti e i componenti elettronici, il primo capitolo tratta le basi della programmazione C e i fondamenti elettrotecnici. Tuttavia, sono spiegate solo le cose più importanti; se si vuole ottenere una conoscenza più approfondita, per esempio nella programmazione, si raccomandano ulteriori letture in materia.

L’approccio a questo libro, quindi, è stato pensato per un lettore con basi tecniche ma senza alcuna conoscenza approfondita di Arduino.

Se pensi di non aver bisogno di recuperare queste aree specifiche, puoi iniziare direttamente con il terzo capitolo che riguarda specificamente la piattaforma Arduino e le diverse versioni dei microcontrollori.

Tuttavia, è consigliabile almeno sfiorare di nuovo le basi.

Per consentire una migor comprensione, troverai le seguenti icone più volte in vari punti del libro:

Simboli aritmetici: Qui il discorso si fa più complesso. Mite questo simbolo si viene introdotti a una digressione o una derivazione. La derivazione di un'area tematica è utile per la comprensione, ma non è essenziale ed è piuttosto intesa come riferimento.

Lampadina: qui vengono riassunti i punti chiave di un capitolo. Queste affermazioni sono buone come riferimento o quando si deve rivedere un'area tematica.

Attenzione: qui sono menzionati gli errori comuni. Viene mostrato dove e perché spesso si incontrano ostacoli o supposizioni sbagliate.

Calcolatrice: esempi di calcoli o domande di comprensione da seguire e interiorizzare.

Ora spero che ti piaccia leggere e che ti piacerà immergerti nell'ecosistema Arduino.

2. Fondamenti di ingegneria elettrica

La piattaforma Arduino combina hardware e software. Per poter utilizzare l'ecosistema Arduino in modo sensato, è necessario comprendere i circuiti e le periferiche sotto forma di componenti. Pertanto, le quantità fisiche di corrente, voltaggio, potenza ed energia sono prima ripetute. Queste informazioni sono note a tutti, ma sono spesso mal utilizzate o interpretate in modo scorretto.

2.1. Il potenziale elettrico φ

Il potenziale elettrico o anche potenziale elettrostatico, la cui unità di misura è il Volt (V) è abbreviato con il simbolo φ (lettera greca minuscola Phi).

Il potenziale elettrico descrive l'energia potenziale di una carica campione all'interno di un campo elettrico. Il campo elettrico assegna un potenziale ad ogni punto dello spazio.

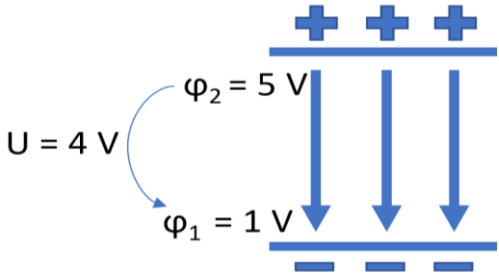


Illustrazione 2 Potenziali e voltaggio nel campo E omogeneo

2.2. Il voltaggio U

La differenza tra due potenziali φ-2 φ, anch'esso con misurato in Volt (V), si 1 chiama voltaggio U.

Bisogna sempre assicurarsi che un voltaggio indichi solo una differenza di potenziale. Pertanto, avrai sempre bisogno di un potenziale di riferimento.

Una batteria normale ha una voltaggio di 1,5 V. Ciò significa che il polo positivo, cioè il punto di contatto superiore della batteria, ha un potenziale elettrico di 1,5 V superiore a quello del polo negativo.

2.3. La corrente I

Una misura della forza del flusso di elettroni è la corrente, abbreviata con il simbolo di formula I. La sua unità di misura è l'ampere (A).

In un circuito tecnico, la corrente scorre sempre dal polo positivo al polo negativo! Anche se gli elettroni fluiscono fisicamente dal polo negativo al polo positivo.

2.4. L'energia E e la potenza P

I termini energia e lavoro sono usati per la stessa quantità fisica. L'energia o il lavoro sono abbreviate con E o W.

2.4.1. L'energia

L'energia indica quanto lavoro può fare un sistema, per esempio quanto calore può essere generato. L'energia è indipendente dal tempo. La sua unità di misura nel SI è il joule [J].

Unità alternative sono i watt-secondi Ws o i chilo-watt-ora kWh. Le unità naturalizzate per l'energia sono, per esempio, le kilo-calorie kcal.

Conservazione dell'energia:

La legge di conservazione dell'energia afferma che c'è una quantità fissa di energia nell'universo che non può essere distrutta, né creata. L'energia può essere convertita solo in forme diverse.

2.4.2. La potenza P

La potenza, d'altra parte, è una quantità fisica e si riferisce all'energia o al lavoro che viene convertito in una certa quantità di tempo.

$$P = \frac{\Delta E}{\Delta t}$$

L'unità di potenza è il watt, che corrisponde all'energia per il tempo. L'unità di misura dell'energia è il joule, mentre l'unità del tempo è il secondo.

Un watt corrisponde quindi a un joule al secondo $1\frac{J}{s} = 1W$

La potenza elettrica convertita in un sistema o componente è calcolata moltiplicando il voltaggio per la corrente.

$$P = U \cdot I$$

2.5. La resistenza R

La resistenza elettrica è, come suggerisce il nome, una resistenza per gli elettroni. È quindi più difficile per gli elettroni fluire attraverso il resistore.

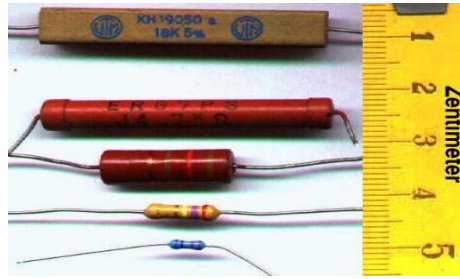


Illustrazione3 Diversi tipi di resistenze

La resistenza elettrica ha il simbolo R e si misura in Ω (Ohm).

La resistenza elettrica indica il voltaggio elettrico U necessario a far fluire un amperaggio I attraverso un conduttore elettrico.

$$\boldsymbol{R = \frac{U}{I}}$$

L'unità Ohm è quindi $1\Omega = \frac{1V}{1A}$

Possiamo riorganizzare le formule di conseguenza per

$$\boldsymbol{U = R \cdot I}$$

Un aiuto che indica la relazione tra voltaggio, resistenza e corrente in un circuito è quindi "URI".

Anche il resistore è un componente elettrico. Quando si parla di un resistore, di solito si intende il componente specifico.

Il simbolo del circuito è disegnato come un rettangolo.

Illustrazione 4 Simbolo del circuito americano *Simbolo del circuito europeo*

2.5.1. Divisore di voltaggio

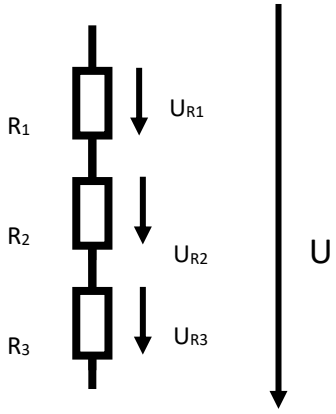


Illustrazione 5 Divisore di voltaggio con collegamento in serie di resistenze

Abbiamo diverse resistenze collegate in serie e conosciamo il voltaggio che passa attraverso tutte le resistenze insieme. Ma quanta voltaggio è passato attraverso le singole resistenze?
Le resistenze dividono il voltaggio tra di loro in proporzione ai loro valori di resistenza, quindi formano un partitore di tensione.

$$\frac{tensione\ alla\ resistenza}{tensione\ totale} = \frac{valore\ della\ resistenza}{resistenza\ totale}$$

$$U_{R1} = U_0 \cdot \frac{R_1}{R_1 + R_2 + R_3 + \cdots}$$

$$U_{R2} = U_0 \cdot \frac{R_2}{R_1 + R_2 + R_3 + \cdots}$$

...

$$U_0 = 10V, R_1 = 6\Omega, R_2 = 4\Omega$$

$$U_{R1} = 10V \cdot \frac{6\Omega}{6\Omega + 4\Omega} = 6V$$

$$U_{R2} = 10V \cdot \frac{4\Omega}{6\Omega + 4\Omega} = 4V$$

2.6. Il diodo

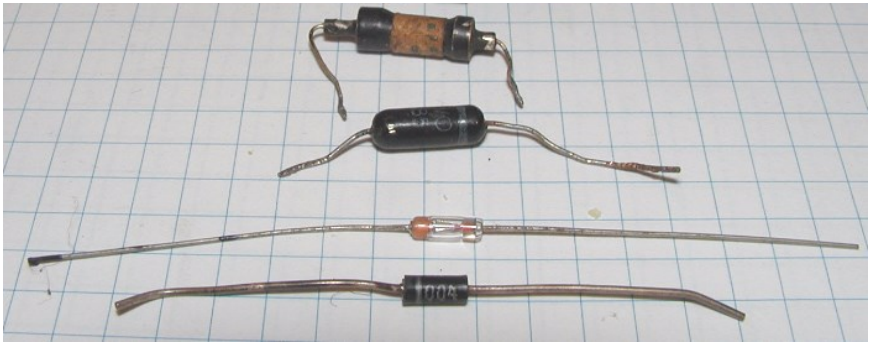


Illustrazione 6 Diversi disegni di diodi

Un diodo è un componente che permette alla corrente di fluire in una sola direzione. Questo è il motivo per cui viene anche chiamato diodo semiconduttore.

Illustrazione 7 Simbolo dell'interruttore Diodo

Il diodo consiste in un anodo collegato al polo positivo o potenziale superioree un catodo, collegato al polo negativo o potenziale inferiore.

Una linea verticale sul componente mostra quale filo rappresenta il catodo.

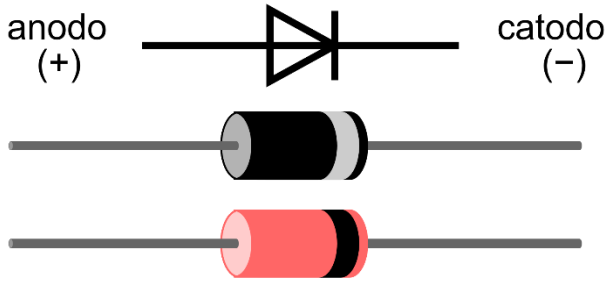


Illustrazione 8 Anodo e catodo di un diodo

Poiché il diodo conduce solo in una direzione, si parla di funzionamento in direzione del flusso o in direzione inversa.

Nella direzione del flusso su un diodo di silicio standard si ha circa 0,7V di caduta di tensione. Con un diodo Schottky, per esempio, solo 0,2 V.

Un diodo non ha quasi nessuna resistenza nella direzione del flusso e quindi non limita il flusso di corrente. Per utilizzarlo avrai bisogno di un'altra resistenza, chiamata resistenza in serie, perché senza di essa c'è il rischio di un corto circuito.

Excursus: LED

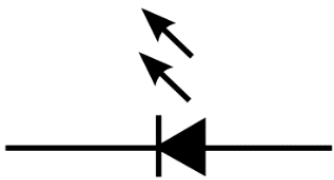


Illustrazione 8.2 Simbolo del circuito LED

Il funzionamento e le proprietà di un diodo ad emissione di luce sono le stesse di un diodo semiconduttore "normale".

Tuttavia, i LED di solito hanno una voltaggio in avanti più alto di circa 1,6 V-2,6 V invece che di 0,7 V.

Un diodo ad emissione di luce non ha quasi nessuna resistenza nella direzione del flusso e quindi non limita il flusso di corrente.

2.7. Il condensatore e la capacità C

Illustrazione 3 Condensatori vari

Simile a una batteria, il condensatore ha la capacità di immagazzinare cariche elettriche, ma solo per periodi di tempo limitati. Pertanto, è perfettamente adatto a sostenereretensione e corrente. Se una fonte di voltaggio non può fornire la quantità di corrente richiesta, il condensatore serve come deposito tampone, per così dire.

Rilascia molta energia in poco tempo quando serve e quando c'è poca carica, si ricaricano.

I condensatori sono quasi sempre installati in parallelo per sostenere le tensioni di alimentazione, per questo motivo sono anche chiamati condensatori di supporto.

La costante che indica il rapporto tra voltaggio e carica si chiama capacità C. La sua unità di misura è il farad F.

In realtà, le capacità dei condensatori sono relativamente piccole. L'ordine di grandezza della capacità è nella gamma di micro-, nano- o picofarad.

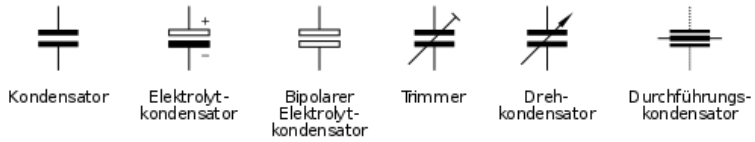


Illustrazione 4 Simboli dei circuiti per diversi tipi di condensatori

2.8. Tecnologia digitale

Dopo aver conosciuto i componenti hardware più rilevanti, veniamo alla tecnologia digitale - un sottocampo dell'ingegneria elettrica. Per capire come la tecnologia digitale ci aiuta nella programmazione, è importante separare le definizioni di "analogico" e "digitale".

Con il termine analogico, si intende un segnale continuo, cioè che comprende un numero teorico infinito di valori intermedi tra due valori.

Esempio cardine del sistema anlogico è l'orologio in cui la lancetta non ticchetta, ma ruota continuamente.

Con un tale orologio, la lancetta può teoricamente assumere qualsiasi posizione all'interno del suo quadrante meccanico e quindi qualsiasi ora.

Una grandezza analogica può assumere qualsiasi valore all'interno dell'intervallo di definizione.

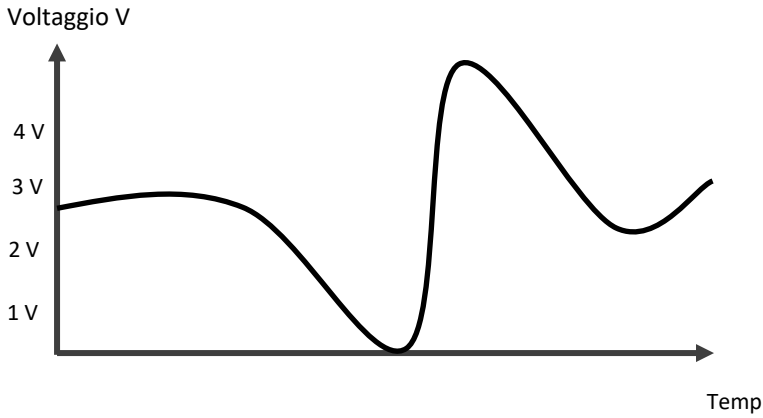


Illustrazione 5 Curva di voltaggio analogica

Questo non è il caso di un orologio digitale. La maggior parte degli orologi digitali senza quadrante - come per esempio le sveglie - non possono nemmeno visualizzare i secondi. L'orologio o la sveglia possono accettare solo valori discreti.

Una quantità fisica che può assumere solo certi valori discreti è chiamata segnale digitale.

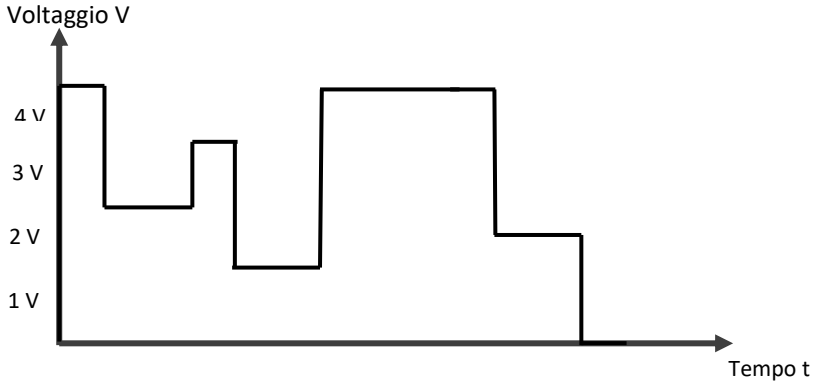


Illustrazione 6 Curva di voltaggio digitale

Curiosità: la parola digitale deriva dal latino *digitus*, che letteralmente significa dito. Questo perché un segnale digitale, proprio come le nostre dita, ha un insieme numerabile di valori. Le calcolatrici, i PC e gli smartphone di oggi usano solo il sistema digitale. Un'unità di memoria, meglio conosciuto come bit, può essere caricato (logico "1") o non caricato (logico "0").

Per poter calcolare una grandezza fisica, per esempio la temperatura esterna, abbiamo bisogno di sensori che convertano la proprietà termica dell'aria in un segnale elettrico.

Un sensore converte una grandezza fisica in un segnale elettrico, in questo caso la temperatura in una voltaggio analogica.

Questo segnale analogico viene poi convertito in un segnale digitale. I convertitori analogico-digitali ADC (analogue-digital-converter) sono utilizzati per questo.

La conversione di un segnale analogico in un segnale digitale si chiama campionamento.

Grazie al segnale digitale, possiamo poi realizzare le nostre solite operazioni aritmetiche (addizione, sottrazione, ...).

Se volete emettere un segnale analogico, per esempio per azionare il motore di un ventilatore, il segnale digitale deve essere convertito in analogico.

La conversione daldigitale in analogico si chiama filtraggio. I convertitori digitale-analogico (DAC) sono utilizzati per questo scopo.

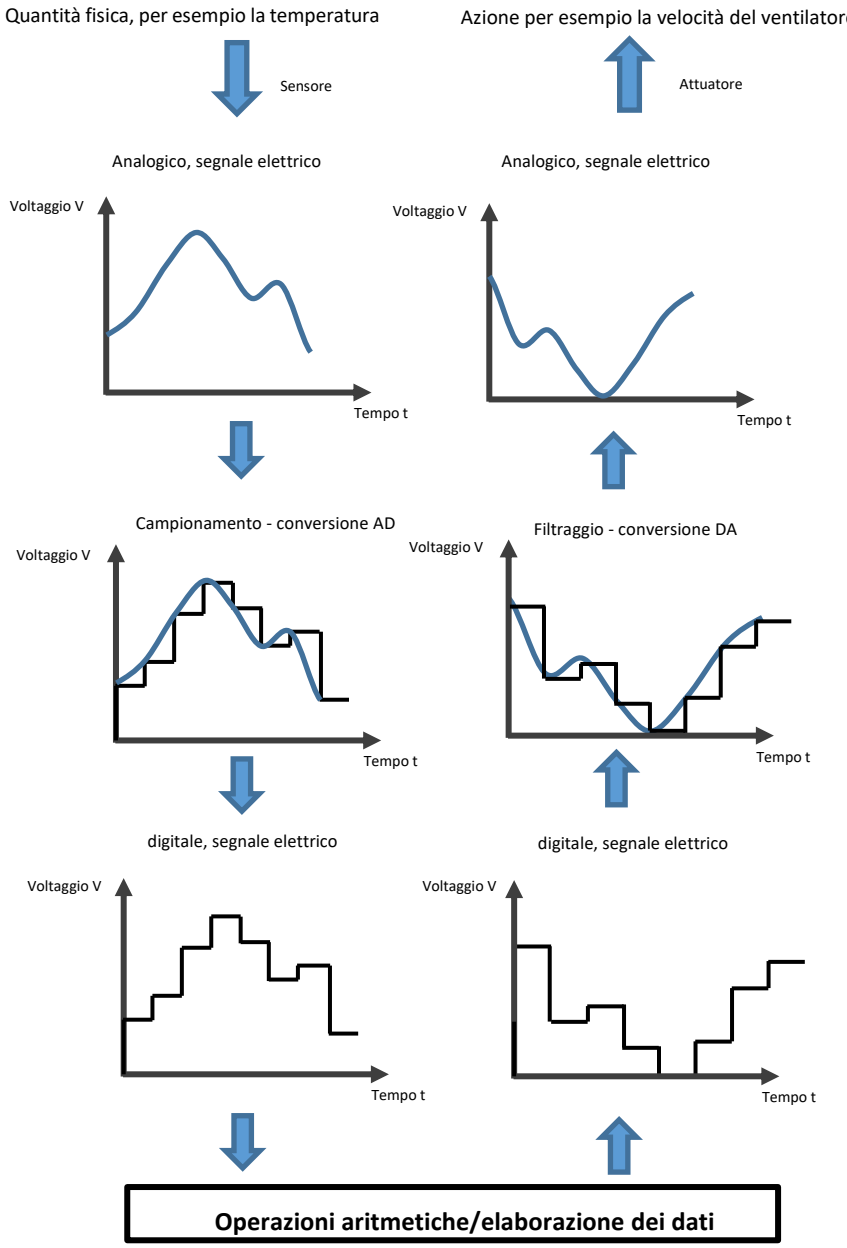


Figura 7 Diagramma di flusso

Bisogna essere consapevoli che c'è una perdita di informazioni durante la conversione analogico-digitale, perché i valori analogici sono "arrotondati" a un valore digitale.

Quando un segnale analogico viene campionato, l'informazione semplicemente si perde. Quanto del segnale sia andato perduto, dipende dalla precisione con cui viene digitalizzato.

Per la temperatura, a seconda dell'applicazione, non è importante se il valore digitale viene emesso in valori interi, cioè a 19 °C, o con una posizione decimale (19,5 °C).

Con immagini, video o file audio, invece, la situazione è diversa. La digitalizzazione durante la registrazione fa perdere informazioni e quindi si ha una minor resa in qualità dei dati. Va detto, tuttavia, che la tecnologia è ormai nella fase in cui questi effetti sono difficilmente percepibili, poiché la scansione è più fine delle nostre soglie di percezione.

2.8.1. Sistemi numerici di un microcontrollore

Questo capitolo è dedicato ai sistemi numerici, grazie ai quali sarà più semplice comprendere come il microcontrollore interpreta I numeri.

Tutti conosciamo il sistema decimale. Il fatto che usiamo 10 cifre è semplicemente perché abbiamo dieci dita e il sistema numerico arabo è stato costruito su di esse.

Contiamo da zero a nove. Dopo di che, abbiamo bisogno di un'ulteriore cifra per poter rappresentare il numero superiore.

Dopo le unità troviamo le decine, le centinaia, le migliaia e così via. Per poterli rappresentare meglio, scriviamo i numeri uno dopo l'altro. Assegniamo un valore più alto alla cifra quanto più avanti si trova all'interno del numero.

La matematica non conosce questa interpretazione. Calcola in base alle potenze. Qui, il 10 costituisce la base, mentre posto dove si trova il numero corrisponde alla potenza.

Le decine corrispondono a 10^1le centinaia 10^2 le migliaia 10^3 ecc.

Subconsciamente, abbiamo usato questa regola per tutta la vita. Per esempio, interpretiamo il numero 1273 come 1000 + 200 + 70 + 3.

Ogni numero può quindi essere scomposto in cifre moltiplicate per una potenza di dieci.

$$1273 = 1000 + 200 + 70 + 3 = 1 \cdot 10^3 + 2 \cdot 10^2 + 7 \cdot 10^1 + 3 \cdot 10^0$$

Ma cosa succederebbe se non prendessimo il 10 come base del nostro sistema numerico, ma il 2? Staremmo parlando del sistema binario, in cui non troveremmo dieci cifre, ma solo due: zero e uno.

Ma allora come rappresentiamo il numero 2? Nello stesso modo in cui rappresentiamo il numero 10 nel sistema decimale - con una cifra in più. Il numero 2è quindi rappresentato come "10".

Allora anche i post non hanno più la ponderazione, $10^0, 10^1, 10^2, \ldots$ ma $2^0, 2^1, 2^2, \ldots.$

Per esempio, il numero 1101 nel sistema binario è interpretato come:

$$1011 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1$$

Nel sistema decimale, il binario 1011 corrisponde quindi all'11.

La seguente tabella mostra la rappresentazione dei numeri nel sistema binario.

Rappresentazione nel sistema decimale	Rappresentazione nel sistema binario
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
...	....
16	10000
32	100000
128	10000000
256	100000000
512	1000000000

Ora sarete in grado di capire la battuta: ci sono solo "10" tipi di persone. Quelli che capiscono il codice binario e quelli che non lo capiscono.

Una cifra è spesso indicata come un bit. Un bit può assumere solo lo stato "1" o "0". Un byte è composto da 8 bit, un byte può quindi assumere la gamma di valori 0000000 (zero decimale) o 11111111 (255 decimale). Questo sarà di grande importanza in seguito, per esempio con i tipi di dati delle variabili.

Se volete che un microcontrollore percepisca un numero come binario, dovrete scrivere una B davanti al numero.

B1011

Perché il sistema binario è così importante per noi? In poche parole, perché il computer non ha dita, ma conosce solo gli stati "voltaggio presente" o "nessuna voltaggio". Tutte le variabili, le operazioni, le funzioni ecc. sono memorizzate al livello più basso come bit e byte.

Ci sono altri sistemi numerici, per esempio il sistema ottale con una base di 8 (segnato con o prima del numero, per esempio o127) o il sistema esadecimale con una base di 16 (segnato con ox prima del numero, per esempio oxE3). Per noi è sufficiente capire il sistema binario.

3. La piattaforma Arduino spiegata

Dopo aver passato in rassegna le basi più importanti dell'elettrotecnica e aver saputo che un microcontrollore lavora con valori digitali, il capitolo seguente spiega cos'è esattamente Arduino, da dove viene il nome e quali sono le diverse versioni.

3.1. Cos'è Arduino

Arduino descrive una piattaforma di calcolo fisico. Questo significa che un sistema interattivo viene creato attraverso l'uso e l'interazione di hardware e software reagendo agli eventi del mondo reale e registrando le variabili analogiche attraverso dei sensori.

Questo significa che anche i profani possono realizzare progetti complessi in modo semplice e rapido.. Soprattutto hobbisti, inventori e armeggiatori di tutto il mondo utilizzano la piattaforma Arduino. Molte aziende, per esempio i produttori di stampanti 3D, usano occasionalmente componenti compatibili con la piattaforma Arduino. Nel frattempo, anche molti studenti stanno imparando ad apprezzare il sistema e, grazie alla facilità di gestione, usano la piattaforma comeintroduzione ai microprocessori.

Nell'uso generale, il termine Arduino è usato sia per le schede, cioè i PCB che includono microcontrollori e periferiche, sia per il software e la programmazione.

3.1.1. Origine

I fondatori della piattaforma Arduino - Massimo Banzi e David Cuartielles - volevano consentire l'accesso alla programmazione dei microcontrollori al maggior numero di persone possibile.

Pertanto, è molto importante che la piattaforma possa essere utilizzata anche da persone meno esperte tecnicamente. Per esempio, la struttura della programmazione è leggermente cambiata. Le librerie complesse sono scritte da programmatori addestrati e l'utente finale può semplicemente usarle senza comprendere l'intera portata della libreria.

Curiosità: Arduino era il nome del bar dove i fondatori si sono incontrati per promuovere la loro idea. Ecco come il nome di una piccola caffetteria in una provincia italiana è diventato un marchio famoso in tutto il mondo con un'enorme comunità.

Inoltre, sia l'hardware che il software sono open source. Questo significa che tutto ciò che circonda il progetto è accessibile liberamente.

Per esempio, gli schemi dei circuiti e i layout delle schede Arduino possono essere scaricati gratuitamente dalla homepage ufficiale di Arduino. D'altra parte, questo rende molto facile copiare la tecnologia, cosa che viene praticata anche da molte aziende cinesi.

3.1.2. Hardware - La scheda Arduino

Rendendo la tecnologia accessibile a tutti, la piattaforma Arduino ha guadagnato rapidamente popolarità. Si sono formate grandi comunità e le aziende tecnologiche stanno producendo sensori e attuatori su misura per l'ecosistema Arduino.

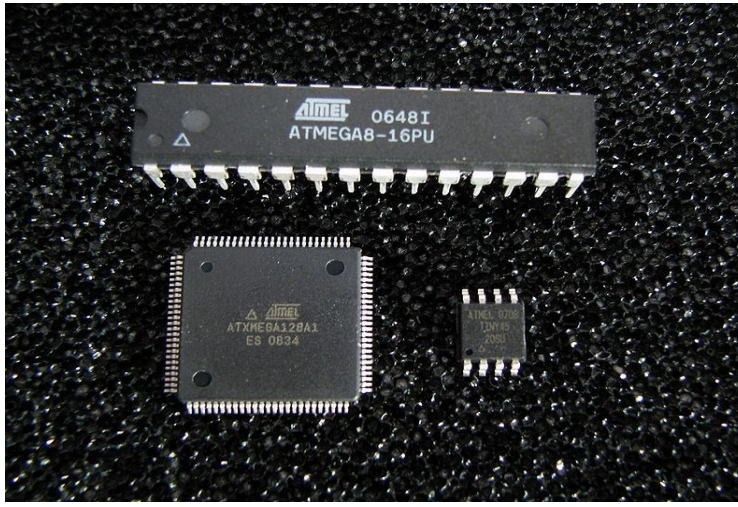


Illustrazione 8 Serie di chip Atmel

L'hardware di solito consiste in un microcontrollore con circuiti ambientali (periferiche). Il microcontrollore usato è della serie Atmel-megaAVR, una famiglia di microcontrollori a 8 bit del produttore di chip Microship negli USA. La famiglia

AVR è stata scelta per la sua facile programmabilità, nella versione con alloggiamento standard dei chip DIP.

Il nome ufficiale Arduino è protetto, ed è per questo che le repliche hanno spesso un nome simile, ma possono essere pubblicizzate solo come "Arduino-compatibili". Esempi di repliche dell'Arduino Uno sono l'Elegoo Uno, il Funduino Uno o la "Sunfounder Board for Arduino Uno".

Le caratteristiche delle varie schede sono più o meno ampie a seconda del modello. I vari modelli di Arduino sono presentati di seguito. Iniziamo con il modello più grande ed esteso, l'Arduino MEGA 2560 R3 e riduciamo passo dopo passo l'estensione fino ad arrivare a un modello con un solo chip senza periferiche, l'Atthiny.

3.2. La famiglia Arduino in sintesi

Il più grande Arduino è l'Arduino Mega 2560. Il nome deriva dal microcontrollore utilizzato, un ATmega2560.

3.2.1. Arduino Mega 2560

Il chip ha 54 pin digitali e 16 analogici e altri pin importanti come i pin GND e i pin per i bus. Arduino ha una presa USB-B attraverso la quale può essere collegato a un PC e programmato. Può essere alimentato tramite questa porta USB o tramite un alimentatore esterno. Maggiori informazioni su questo nel capitolo "Cinque modi di alimentazione".

Arduino Mega 2560 è stato ideato per grandi progetti, specialmente grazie alla sua grande memoria interna e ai numerosi ingressi e uscite.

Oggi, molte stampanti 3D sono basate su un chip ATmega2560.

Con uno schermo adatto (vedi capitolo 3.5), un normale Mega 2560 può anche diventare la scheda madre di una stampante 3D. Il software Marlin è disponibile gratuitamente.

3.2.2. Arduino Uno

Arduino Uno è il modello intermedio della famiglia Arduino.

Viene fornito su un circuito con strisce di prese saldate. Tutti i pin importanti, come gli ingressi e le uscite analogiche e digitali, sono presenti, così come vari bus, ad esempio per controllare una scheda SD.

Rispetto all'Arduino MEGA 2560, ci sono significativamente meno pin. Tuttavia, di solito non avete bisogno di tutti i pin di Arduino Uno.

Inoltre, Arduino Uno costa solo poco più della metà di un Arduino Mega. È anche molto popolare, quindi c'è una grande comunità di utenti che lo utilizza, consentendo una maggior diffusione di accessori ed estensioni adatte.

Arduino Uno può essere descritto come un tuttofare nella famiglia Arduino ed è ideale per i principianti. Ecco perché è questo il modello che utilizziamo in questo libro per presentare i progetti.

3.2.3. Arduino Nano

Esternamente, Arduino Nano è molto diverso dai suoi fratelli maggiori. La presa USB-B è sostituita da una presa mini-USB. Anche la presa di collegamento per l'alimentazione esterna è stata rimossa. Questo è anche il suo più grande vantaggio. Utilizza lo stesso microcontrollore di Arduino Uno, ma richiede solo un quarto dello spazio della scheda.

Arduino Nano è l'ideale per progetti in cui il risparmio di spazio è importante ma non si vuole scendere a compromessi sulla funzione.

3.2.4. Arduino Pro Mini

Se lo vuoi ancora più piccolo, puoi ripiegare su Arduino Pro Mini. Rispetto ad Arduino Nano, Arduino Pro Mini non ha una porta USB, quindi l'area della scheda può essere rimossa. Avrai quindi bisogno di un adattatore esterno per la programmazione.

3.2.5. ATthiny

La famiglia ATthiny non fa direttamente parte della famiglia Arduino. Dal momento che i controller possono essere programmati tramite l'IDE di Arduino, sono menzionati per mostrare come può essere impostato un progetto Arduino

minimalista. I chip ATthiny sono microcontrollori "nudi" senza periferiche e richiedono lo sviluppo di un circuito separato per l'uso.

Un Arduino Uno, per esempio, può essere usato per la programmazione di Atthiny.

Oltre ai disegni presentati, ci sono anche diverse varianti e casi speciali per diverse aree di applicazione.

Dato che la costruzione degli Arduino è open source, ci sono anche molte copie da comprare dai produttori dell'Estremo Oriente. Questi di solito usano componenti hardware molto più economici, il che aumenta la probabilità di fallimento.

Dopo aver appreso le caratteristiche delle diverse versioni, veniamo alla struttura del circuito di un Arduino. Qui ci occuperemo di Arduino Uno, dato che è il modello standard. Offre un mix di possibilità di connessione e facilità d'uso per i principianti.

3.3. Struttura di Arduino Uno

La scheda ha molti componenti intorno al microcontrollore, come l'alimentazione e le porte USB per la programmazione tramite comunicazione con il PC.

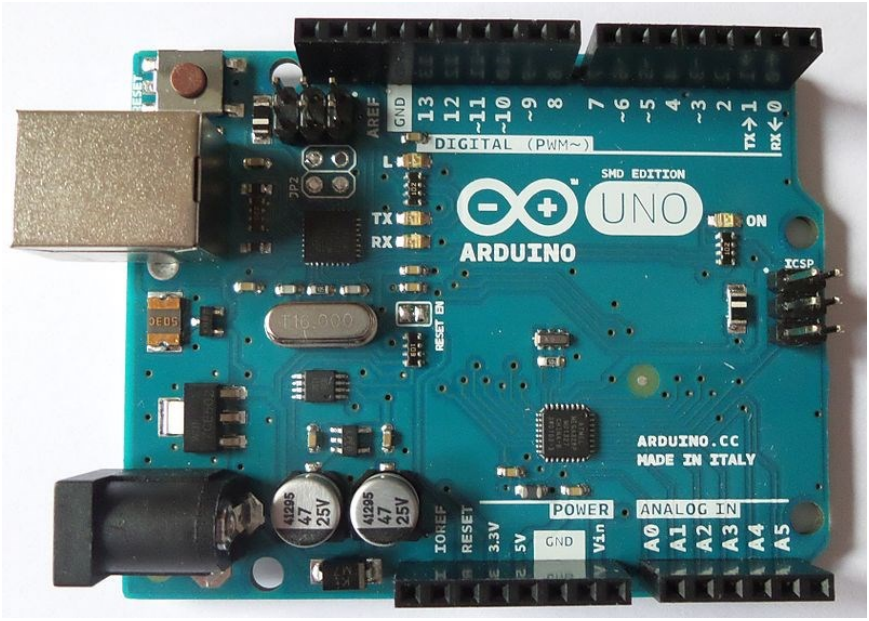


Illustrazione 9 Arduino UNO visto dall'alto

3.3.1. Connessioni e interfacce

Cominciamo con l'ovvio, i pin e le prese.

Presa USB: La presa USB di tipo B può essere usata per l'alimentazione e per la comunicazione con un computer. Per l'alimentazione, assicurati che la porta USB esterna fornisca sufficiente potenza.

La presa a bassa voltaggio è usata solo per l'alimentazione. Ha una gamma di voltaggio d'ingresso di 7-12 V DC. DC sta per corrente continua.

Di nuovo, assicuratevi di usare un alimentatore con una potenza sufficiente.

Nella zona superiore e inferiore, c'è una fila di faggi con perni per uso digitale. I pin TX e RX sono collegamenti per una connessione di bus seriale. I pin digitali 2 - 13 possono essere usati come ingressi o come uscite. I pin possono accettare sia il valore 0 V che 5 V. I livelli intermedi non sono accettati. La tilde (~) accanto ai pin significa che questi pin possono essere usati come un segnale PWM

(segnale di modulazione di larghezza di impulso), ad esempio per controllare un motore. Con questi pin si può generare un voltaggio tra lo 0 e i 5 V.

Troverai un pin di terra (GND), un pin per un voltaggio di riferimento analogico (AREF) e due pin I2C (SCL e SDA) per una connessione al bus I²C. Accanto c'è un pulsante di reset, per resettare il microcontrollore senza interrompere la connessione all'alimentazione.

La sezione inferiore contiene i pin analogici A0-A5. Questi possono anche essere usati come ingressi e accettano valori da 0 V a 5 V compresi di stadi intermedi.

Come può Arduino calcolare con un voltaggio analogico? Quali componenti sono necessari per questo?

Soluzione: Il voltaggio analogico al pin viene prima digitalizzato con l'aiuto di un convertitore AD. Arduino è dotato di un convertitore AD a 10 bit integrato adatto allo scopo.

La maggior parte dei pin, sia digitali che analogici, sono indirizzati direttamente al microcontrollore, il che ci porta direttamente al cuore della questione, l'ATMEGA328P.

L'ARMEGA328P è il cervello di Arduino. Il chip è disponibile in due versioni, la versione plug-and-play con una presa sulla scheda e la tecnologia SMD (surface mounted device) piatta. Anche se visivamente diverso, è lo stesso chip con le stesse proprietà e opzioni di connessione.

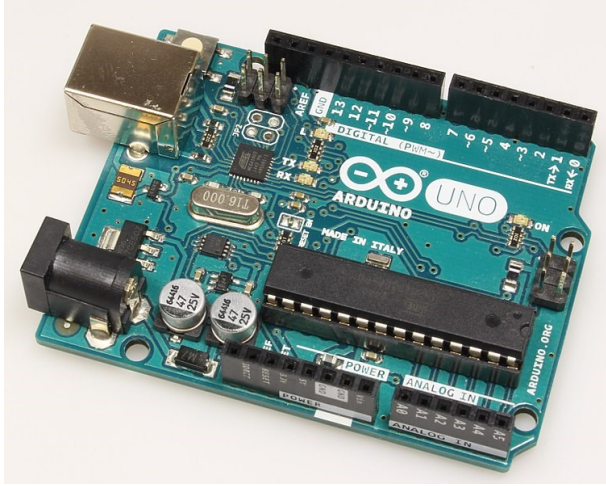


Illustrazione 10 Arduino UNO con chip collegabile

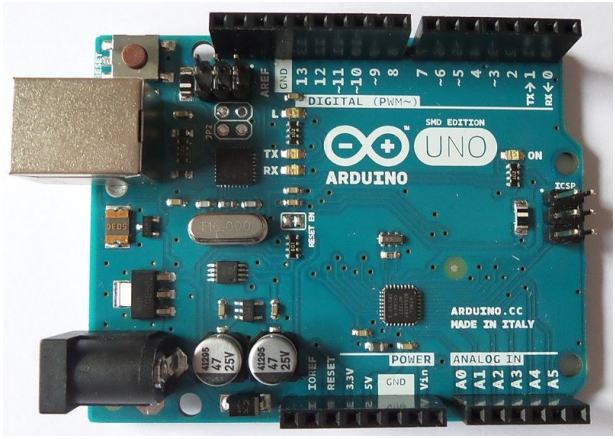


Illustrazione 11 Arduino UNO con chip SMD fisso

Oltre al chip principale, è installato un altro chip, l'ATMEGA16U2. Questo è montato direttamente accanto alla presa USB e converte i segnali USB in segnali seriali che l'ATMEGA328P può capire. Praticamente funziona come un "traduttore" tra il PC e l'ATMEGA328P.

Ci sono anche due LED sulle linee dati Tx e Rx. Quando questi lampeggiano, indica che la connessione seriale sta funzionando. Questo accade, per esempio, quando il microcontrollore riceve dati di programmazione o di programma tramite il PC.

Oltre a questi LED, c'è un altro diodo che indica il voltaggio di funzionamento (etichetta ON) e un LED (L) che è collegato al pin digitale 13, utile, adesempio, per verificare se il programma sta controllando il PIN durante il suo impiego.

I pin che si trovano più volte sulla scheda sono direttamente collegati tra loro. Questi includono, per esempio, i pin GND o i pin delle interfacce dati (SDA, SCL).

I componenti rimanenti, come i due grandi condensatori e le resistenze, sono utilizzati per la stabilizzazione del voltaggio o come circuito di protezione. Il cristallo ovale da 16 MHz sincronizza il microcontrollore.

Infine, c'è un componente che è responsabile dell'alimentazione. Genera un voltaggio costante di 5V DC per il microcontrollore e 3,3V per le applicazioni esterne dalvoltaggio di ingresso nell'intervallo 7-12V. Tuttavia, qui c'è da tenere a mente una cosa importante, altrimenti potrebbe accadere che il convertitore di voltaggio o un dispositivo esterno possano dirvi addio.

Come già detto, il voltaggio d'ingresso della presa a bassa voltaggio è alimentato al regolatore di voltaggio. In alternativa, potete anche alimentare il voltaggio direttamente tramite il pin Vin.

Il pin Vin è collegato alla presa di basso voltaggio.

Molti adattatori o estensioni sono progettati per 5 V o 3,3 V. Pertanto, potete collegarli al pin Vin solo se usate un'alimentazione a 5 V o un'alimentazione via USB. Altrimenti, il pin 5 V deve essere selezionato.

Quindi, prima di collegare un'alimentazione ad Arduino, è importante controllare se le prolunghe sono collegate al pin Vin.

Inoltre, bisogna notare che il convertitore di voltaggio a 5 V è progettato solo per correnti molto piccole. Secondo la scheda tecnica, può fornire un massimo di 200 mA, cioè 1 W di potenza. Quando si azionano diversi motori, per esempio, si deve sempre collegare un'alimentazione esterna perché altrimenti il regolatore di voltaggio sarà sovraccaricato.

3.3.2. Pinout

Lo schema della piedinatura è importante per controllare correttamente i pin nel programma. Ogni perno ha una o più denominazioni.

I nomi dei pin si trovano scritti anche sulla oarte laterale di Arduino. Tuttavia, a volte la necessità di programmare ci assale anche quando Arduino non è a portata di mano. È quindi consigliabile tenere bene il segno con un segnalibro.

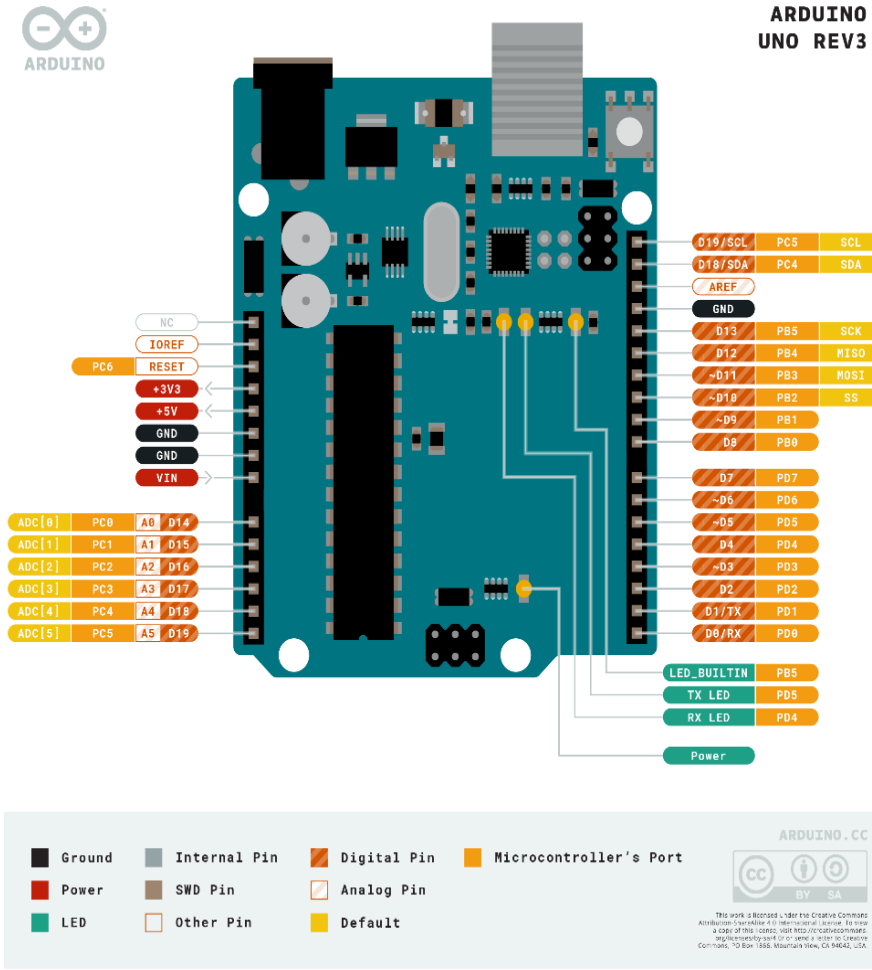


Illustrazione 12 Pinout di Arduino UNO

I pin analogici A0-A5 possono anche essere usati come pin digitali D14-D19 se necessario.

3.4. Altro hardware

La piattaforma Arduino ha guadagnato rapidamente popolarità. Molte aziende hanno visto le potenzialità di Arduio sul mercato e si sono concentrate su estensioni e componenti adatti alla piattaforma. Di seguito, verranno presentati alcuni dei più importanti sensori, attuatori e schede.

3.5. Scudi Arduino

Gli Scudi Arduino sono schede di espansione che si adattano precisamente alle dimensioni di un Arduino. Le dimensioni e le assegnazioni dei pin sono progettate in modo tale da corrispondere ai pin di Arduino e possono essere posizionate direttamente su Arduino.

Gli scudi Arduino sono schede di espansione personalizzate progettate per un obiettivo specifico.

Le schermature stesse spesso contengono componenti elettronici aggiuntivi che sono destinati a un'area specifica di applicazione. Un esempio è il Motorshield. Si adatta a un Arduino Uno e Arduino MEGA2560. I driver del motore, le periferiche e le prese per collegare i motori DC sono installati sulloscudo. In questo modo, un Arduino può diventare il centro di controllo di un automodello.

Con l'aiuto di questi scudi, anche i progetti più grandi possono essere realizzati facilmente.

Per esempio, con uno shield Ramps 1.6 e il codice del programma corrispondente, un Arduino MEGA2560 diventa un'unità di calcolo completa per una stampante 3D.

In relazione a tali schermi, di solito sono necessari sensori e/o attuatori.

3.6. Sensori e attuatori

Nel capitolo 2.8 abbiamo già imparato che i sensori generano un segnale elettrico, per lo più analogico, da una grandezza fisica.

In alternativa, ci sono anche sensori con elettronica di valutazione integrata che possono misurare la quantità fisica e trasmettere i risultati della misurazione ad Arduino tramite un'interfaccia. L'interfaccia può essere realizzata tramite uno (o più) cavi di dati. Tutti i sensori disponibili hanno sia vantaggi che svantaggi. A seconda del tipo di sensore che si utilizza ci sarà una diversa interrogazione da effettuare per ottenere i dati. Poiché i sensori sono una grande parte dell'ecosistema Arduino, impareremo più tardi come controllarne cinque.

. Adesso troveremo un termine meno comune di sensore: attuatore.

Gli attuatori sono la controparte dei sensori. Convertono un segnale elettrico in un cambiamento di una quantità fisica. Per esempio, un motore elettrico converte il segnale di uscita in movimento meccanico, o un elemento riscaldante converte il segnale di uscita in calore.

3.7. Breadboard e cavo (jumper)

Per poter costruire rapidamente un circuito, abbiamo bisogno di una breadboard.

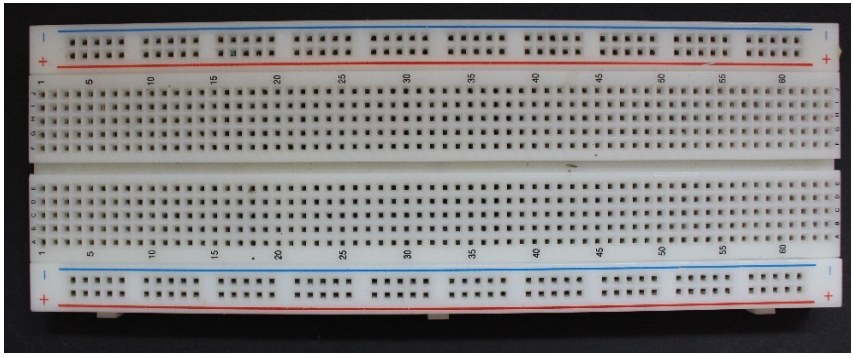


Illustrazione 13 Pegboard

Una breadboard è composta da molte file e colonne. Le file interne sono collegate tra loro e l'alimentazione è collegata all'esterno. L'illustrazione seguente mostra quali contatti della breadboard sono collegati tra loro.

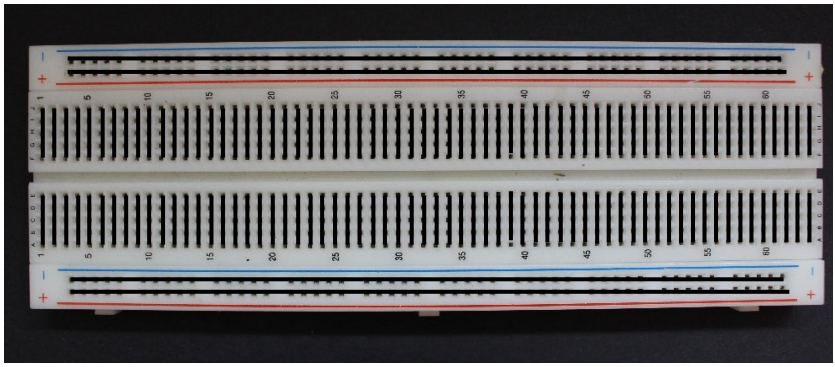


Illustrazione 14 Scheda plug-in con connessioni disegnate

Le lavagne a pioli sono disponibili in molte dimensioni e design. I collegamenti sono quasi sempre identici. Ogni set di Arduino per principianti viene fornito con una breadboard di questo tipo. Per lo più in relazione ai cavi di collegamento.
Invece di saldare i cavi insieme, usiamo cavi di collegamento e connettori. I cavi jumper hanno un perno alla loro estremità che si inserisce nelle prese di Arduino e della scheda plug-in.

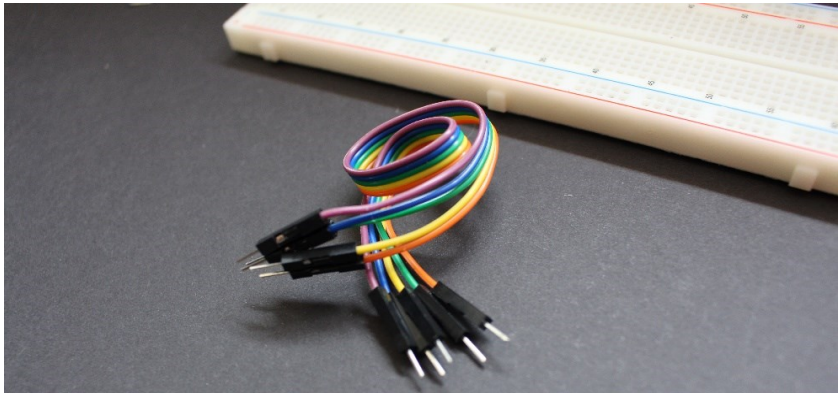


Illustrazione 15 Cavo jumper

I connettori, d'altra parte, sono progettati esclusivamente per effettuare connessioni sulla breadboard.

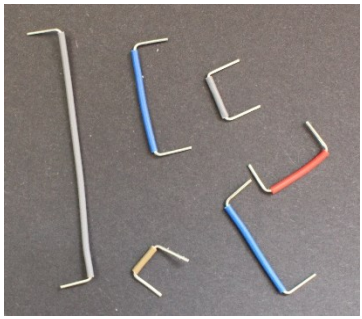


Illustrazione 16 Connettore

Grazie al semplice meccanismo a innesto, è possibile costruire rapidamente dei circuiti. Naturalmente, assicuratevi sempre quali file sono collegate!

Per I principianti sono altamente raccomandati gli starter kit che potri trovare da vari produttori in diverse dimensioni. A seconda dello scopo, contengono LED, sensori, motori, display e molti altri componenti. Una breadboard e dei cavi fanno parte dell'attrezzatura di base.

4. Basi della programmazione

Di seguito ti spiegheremo le basi della programmazione C.Naturalmente, un'introduzione completa e onnicomprensiva non può essere spiegata in un solo capitolo. L'obiettivo è piuttosto quello di imparare la logica e la struttura di un programma C nella programmazione di Arduino.

Per dare vita a un microcontrollore, bisogna riempire le celle di memoria vuote con del codice preciso. Ciò vale non solo per la famiglia Arduino, ma per qualsiasi sistema elettronico controllato da chip. Una cella di memoria è chiamata bit. La cella può essere riempita eavere quindi valore di "1", o può essere vuota, ed essere rappresentata da "0". Bisogna tenere a mente anche il byte unitario, che corrisponde esattamente a 8 bit.

Questi bit e byte ancora vuoti vengono riempiti dalla programmazione. A questo scopo va creato un codisce specifico.

A questo punto, un programma (codice sorgente) non è altro che un testo multilinea.

Il codice di un programma può essere aperto e modificato con normali editor di testo, senza la necessità di un programma specifico.

Questo testo viene poi interpretato da un compilatore che traduce il codice del programma nel codice macchina trasmettendolo succesivamente al microcontrollore.

Per creare il codice, avete bisogno di un linguaggio di programmazione adatto che il compilatore sia in grado di interpretare.

Ogni linguaggio di programmazione usa la propria sintassi, punteggiatura e parola segnale. Se usate un linguaggio di programmazione diverso da quello previsto, il compilatore non può "capire"/interpretare il codice.

Il microcontrollore usato su un Arduino può essere programmato tramite la sua interfaccia (Arduino IDE). Si possono usare varianti dei linguaggi di programmazione C, C++ e Java.

Il fatto che si possa usare Java è particolarmente vantaggioso per i programmatori che hanno già lavorato su interfacce grafiche con Java, linguaggio spesso usato per le applicazioni web in passato.

Per la maggior parte, comunque, gli Arduino sono programmati in C, e occasionalmente in C++. Quasi ogni tutorial, ogni manuale e ogni libro di riferimento sull'argomento Arduino è basato sul linguaggio di programmazione C.

La programmazione con C è quasi illimitata. È adatta ai principianti e facile da capire. Di seguito troviamo gli argomenti essenziali per la programmazione di Arduino.

Tutte le seguenti informazioni si riferiscono alla programmazione di Arduino con C. Per la maggior parte, queste sono anche informazioni generali sulla programmazione C, ma ci sono piccole particolarità tipiche di Arduino.

4.1. Commenti e istruzioni

I commenti servono alla chiarezza e alla struttura di un programma. Un commento non viene tradotto dal compilatore, viene usato solo dagli esseri umani per capire meglio il programma.

4.1.1. Commenti

I commenti servono alla chiarezza e alla struttura di un programma. Un commento non viene tradotto dal compilatore, viene usato solo dagli esseri umani per capire meglio il programma.

L'intestazione dei commenti viene realizzata con due barre.

```
// questo è un commento
```

Se vuoi commentare più linee in una volta, inizia con una barra seguita da un asterisco. Allo stesso modo, chiudi la sezione dei commenti con un asterisco seguito da una barra.

```
/* questo è un commento

questo qui è ancora un commento

anche questo è un altro commento */
```

I commenti sono importantissimi, soprattutto quando diverse persone lavorano su un progetto o hanno bisogno di capire lo stesso codice.

Inoltre, all'inizio di una programmazione, è consigliabile includere un commento di intestazione che contiene tutti i dati importanti sul programma. Per esempio:

```
/* Codice del programma: Esempio di progetto Arduino

Autore: Max Mustermann

Creato: 12.12.2021

Stato del progetto: 01.01.2022

*/
```

4.1.2. Istruzioni

Le istruzioni costituiscono il nucleo di un programma. Qui, per esempio, si possono trovare operazioni aritmetiche e comandi di input o output. Anche definire variabili, chiamare funzioni o interrompere un ciclo è un'istruzione.

Una dichiarazione si conclude con un punto e virgola. In questo modo, il compilatore sa che il comando precedente deve essere interpretato come una dichiarazione.

```
Istruzione1; //Commento sull'istruzione1

Dichiarazione2;

Dichiarazione3;
```

4.2. Variabili

Le variabili sono numeri che possono essere cambiati durante l'esecuzione di un programma. Sono utilizzati per memorizzare valori o numeri,devono essere inizializzate all'inizio di un programma e possono essere cambiate durante l'esecuzione. La denominazione delle variabili dovrebbe essere significativa e breve. Per esempio, una variabile in cui viene memorizzato un valore di temperatura potrebbe essere chiamata Temp1.

Se ci sono diverse variabili simili, è consigliabile distinguerle per nome o per indici.

4.2.1. Tipi di dati

In C, ci sono diversi tipi di variabili che differiscono per dimensione (requisiti di memoria) o contenuto.

Per esempio, se una variabile può diventare solo 0 o 1, un solo bit di memoria è sufficiente per mappare l'informazione. Non ha quindi senso assegnare 2, 3 o 10 bit alla variabile. Ha sempre senso programmare con parsimonia. All'inizio, non ci sono problemi, ma con l'esperienza, anche i progetti diventano maggiori e quindi sarà molto semplice esaurire rapidamente la memoria se si "spende" la memoria massima per ogni variabile.

L'esempio con la variabile è chiaramente una semplificazione. Naturalmente, ci sono metodi molto più complessi perconservare memoria.

Un esempio di programmazione subottimale è l'interfaccia utente di Windows. Tutti conoscono Windows o l'hanno certamente usato in qualche momento della loro vita. Anche Windows è, ovviamente, stato programmato. Naturalmente, il codice sorgente di Windows è ormai gigantesco e non può più essere tenuto sotto controllo dalle singole persone.

Tuttavia, gran parte del codice non è essenziale, è superfluo. A causa della complessità e della dimensione del codice, tuttavia, nemmeno i programmatori oggi responsabili sono in grado di dire se parti del codice possono essere rimosse senza avere un impatto altrove. Seguendo il motto "non cambiare mai un sistema in funzione", Windows è quindi molto più grande di quanto sia necessario,

con quasi 60 GB di memoria richiesta. La seguente tabella elenca i tipi di variabili più importanti con la loro gamma di valori. Ci sono innumerevoli altri tipi di variabili, ma sono di poca o nessuna importanza per i neofiti.

Nome	Abbreviazione	Requisiti di stoccaggio	Gamma di valori	Caratteristica speciale
Booleano	booleano	1 bit	0 e 1 (vero o falso)	
Byte	byte	8 bit = 1 byte	0...255	Nessun valore negativo
Carattere	char	8 bit = 1 byte	-128...127	Salva una lettera
Carattere senza segno	unsigned char	8 bit = 1 byte	0...255	Nessun valore negativo
Integro	int	16 bit = 2 byte	-32.768...32.767	Solo numeri interi
Intero senza segno	unsigned int	16 bit = 2 byte	0...65535	Solo numeri interi
Lungo	lungo	32 bit = 4 byte	-2.147.483.648 ... 2.147.483.647	
Unsigned long	unsigned long	32 bit = 4 byte	0 .... 4.294.967.295	Nessun valore negativo
Galleggiante	float	32 bit = 4 byte	$-3{,}4028235 \cdot 10^{38}$... $3{,}4028235 \cdot 10^{38}$	Numeri con virgola (fluttuanti)

Vogliamo misurare e memorizzare l'altezza di un bambino nel corso della sua vita. L'altezza è usata in cm e memorizzata in una variabile.

Quale tipo di dati è più adatto per memorizzare la dimensione?

Soluzione: sarebbe il più sensato utilizzare i dati in byte. Con questo tipo di dati si possono memorizzare valori da 0 cm a 255, nel nostro esempio centimetri (255 cm = 2,55 m). Anche il tipo di dati char funzionerebbe, ma questa tipologia si usa per memorizzare lettere e non numeri. Altri tipi di dati come int, long o float potrebbero comunque funzionare bene, ma occuperebbero molta più memoria.

Oltre alle variabili, cioè i numeri variabili, ci sono anche le costanti cherimangono, come suggerisce il loro nome, invariate.

Per generare una costante, scriviamo il termine const davanti al tipo di dati che intendiamo usare.Questoconsente di capire,, soprattutto al lettore del programma, che si tratta di una costante.

In alternativa, si può usare anche il comando #define. In questo caso, il compilatore sostituisce il nome della costante con il valore della costante (qui 6) in tutti i punti in cui la costante compare.

```
#define example constant 6
```

Oppure

```
const int Esempio costante = 6;
```

Per i principianti, entrambe le varianti sono equivalenti, per lo più si usa const.

4.2.2. Operatori

Con le variabili è possibile fare dei calcoli o confrontarne diverse tra loro. Questo sarà molto importante nella prossima sezione: funzioni e cicli che ci consentirà di approcciare agli operatori principali.Le variabili sono retaggi che ci portiamo dallo studio della matematica, qui, invece intendiamo spiegare le peculiarità della loro sintassi in C. È importante capire che il risultato di un confronto può portare a "vero" o "falso". Per far questo, però, dobbiamo conoscere perfettamente le regole che ci chiariscano il quando e il come.

Operatore	Significato	Esempio	Funzione
	Operatori aritmetici		
=	Assegnazione	X=3;	
+	Aggiunta	x=Y+2;	
++	Incremento	x++;	X è aumentato di uno Notazione abbreviata per x=x+1;
-	Sottrazione	x=Y-2;	
--	Decremento	x--;	X si riduce di uno Notazione abbreviata per x=x-1;
*	Moltiplicazione	x=y*2	
/	Divisione	x=y/2	
%	Modulo	X= 10%3;	Fornisce il resto quando si divide. 10 diviso per 3 dà 3, resto 1, quindi x assume il valore 1.

Operatori di confronto			
<	Più piccolo?	3<5	
<=	Più piccolo uguale?	3<=5	
>	Più grande?	5>3	
>=	Più grande uguale?	5>=3	
==	In un momento?	5==4	
!=	Ineguale?	4!=5	
Operatori logici			
!	Negazione	x=!y;	Se y è vero, x diventa falso e viceversa
&&	E	x= 1&&0	vero corrisponde a vero, zero corrisponde a falso, quindi x è anche falso (perché entrambi devono essere veri
||	Oppure	x=1||0	true corrisponde a vero, zero corrisponde a falso, quindi x è vero

4.3. Funzioni

. Anche le funzioni sono reminescenze degli anni passati sui libri di scuola. y = mx+c è il classico esempio di una funzione lineare. L'idea di base è che si esegue una funzione sul numero x ottenendo infine un numero y.

Tuttavia,nella programmazione una funzione non si limita ai valori di input o output come numeri.

Per esempio, potrai utilizzare una lettera o una parola intera invece di un numero non ottenendo necessariamente indietro un numero.

Come "feedback" di una funzione, ad esempio, un motore può ruotare, un'immagine o un testo può essere visualizzato.

Come esempio prendiamo la funzione analogRead(), discussa più in dettaglio nel capitolo 5.3.4;

La funzione legge il valore di voltaggio che viene applicato a un pin di ingresso analogico di Arduino.

Come parametro di ingresso, alla funzione viene detto quale pin di Arduino deve essere letto. Il valore di ritorno corrisponde al voltaggio sul pin coinvolto.

Zero sta per 0 V e 1023 sta per 5 V.

Esempio:

```
temperatura int;

Temperatura = analogRead(A8);
```

Alla variabile "temperatura" viene assegnato il valore che viene applicato al pin A8.

4.4. Cicli e condizioni

I cicli e le condizioni sono strumenti ausiliari per controllare un programma in modo più chiaro ed efficiente. I loop possono essere usati per ripetere certe sezioni del programma e le condizioni possono essere usate per saltare o eseguire parti del programma sotto certe condizioni.

Una delle dichiarazioni più importanti quando si tratta di programmazione è l'istruzione if.

4.4.1. If-else-statement

L'istruzione "if-then" viene eseguita solo se una condizione è soddisfatta. La condizione è scritta tra parentesi tonde dopo la parola segnale "if". È importante capire che la condizione è " controlla se l'affermazione è vera. ". Vedere anche la tabella degli operatori 4.2.2.

Tutte le istruzioni che vengono eseguite quando la condizione è soddisfatta sono scritte tra parentesi graffe.

```
if(condizione){

istruzione1;
istruzione2;

....
}
```

Se la condizione è soddisfatta, il codice del programma tra le parentesi graffe viene eseguito.

Se la condizione non è soddisfatta, l'istruzione If viene saltata e il programma continua dopo le parentesi graffe.

Se vuoieseguire dichiarazioni alternative, cioè se la condizione non è soddisfatta, usa la dichiarazione else, che è opzionale e può essere omessa.

Esempio:

```
int Temperatura;                    // Variabile per la temperatura

String Stato;                       // Variabile per lo stato

if(temp>24){                  // Solo se Temp > 24°C...

Stato = "è troppo caldo";     //...è troppo caldo

}

else{

Status="non è troppo caldo";
```

```
}
```

Solo se il valore della temperatura sale oltre 24, alla variabile viene assegnato lo stato "troppo caldo".

Se non viene usato nessun operatore di confronto, ogni numero non uguale a zero viene interpretato come "Vero". Solo lo zero stesso viene interpretato come "Falso".

Controllate voi stessi, quando viene soddisfatta la condizione if? Quando non lo è?

```
if(3>5) { ..... }

if(3<5) { ..... }

if(3) { ..... }

if(0) { ..... }

if(!3) { ..... }

if(2==2) { ..... }

if( (4>2) && (3<1) ) { ..... }
```

Soluzione:

1. No, 3 è più piccolo di 5, quindi l'istruzione if non viene eseguita.
2. Sì, 3 è meno di 5, quindi l'istruzione if viene eseguita.
3. Sì, 3 è un numero non zero, quindi viene interpretato come "vero" e l'istruzione if viene eseguita.
4. No, il numero zero viene interpretato come "falso" e l'istruzione if non viene eseguita.
5. No, il numero tre viene interpretato come vero ma negato, quindi l'istruzione if non viene eseguita.

6. Corretto, perché 2==2 è un risultato vero.

Attenzione, qui "==" deve essere usato per controllare un'affermazione vera. L'operatore "=" è un'assegnazione, non un controllo, (2=2) risulterebbe in un errore.Tranquillo, è un errore molto comune!

7. No, 4 deve essere maggiore di 2 e 3 deve essere minore di 1, quindi l'istruzione if non viene eseguita.

4.4.2. For-Statement

La dichiarazione for ha una struttura più complessa.

L'istruzione for ripete le dichiarazioni scritte tra parentesi graffe finché la condizione è soddisfatta. Spesso viene definita una variabile contatore dato che il conteggio va in alto o in basso.

Struttura:

```
for(inizializzazione della variabile di conteggio; condizione; incremento) {

        }
```

Esempio:

```
for(int i=0; i <= 10; i++){

  // L'istruzione viene eseguita un totale di 11 volte

 }
```

L'istruzione in questo caso viene eseguita 11 volte perché la variabile di conteggio inizia da i=0 e la condizione di test è i<= 10. Se fosse i<10, il ciclo verrebbe eseguito solo 10 volte.

4.4.3. While

Un ciclo while continua a funzionare finché la condizione racchiusa tra le parentesi tonde non diventa "falsa". Il ciclo può anche funzionare indefinitamente se la condizione non diventa mai "falsa" secondo la logica booleana.

Un ciclo While funziona finché la condizione del ciclo è “vera”. Se nel ciclo while non viene eseguito nulla che influenzi la condizione, il processo si esegue a tempo indefinito.

Esempio di un ciclo che viene eseguito 10 volte.

```
int i = 0;                    //contatore variabile

while(i < 10) {

  i++;                                  // aumenta i di 1

}
```

4.4.4. Do-While

Il ciclo do-while funziona in modo molto simile al ciclo while. La grande differenza è che la condizione viene controllata solo alla fine del ciclo, che viene eseguito almeno una volta.

Esempio:

```
int i = 0;                    //contatore variabile

do {

 i++;                                  // aumenta i di 1

} while(i < 10)
```

4.4.5. Loop()

Il loop() è un loop infinito tipico di Arduino. È necessario affinché il microcontrollore abbia sempre qualcosa da fare. Si può pensare al ciclo come alla "modalità di riposo" del microcontrollore. È più o meno un ciclo while che è sempre "vero".

4.5. Librerie

Le librerie sono un altro enorme vantaggio del concetto open-source di Arduino. Una libreria non è altro che un’implementazione virtuale della funzionalità del software Arduino che aggiunge più comandi e funzioni.

Dietro questo ci sono di solito enormi codici la cui comprensione della struttura non è necessaria per l'uso.

Per esempio, è molto noioso programmare a mano il controllo di un servomotore.

Invece, c'è una libreria già pronta attraverso la quale è possibile controllare i servimotore usando semplici funzioni. Dopo tutto, perché dovremmo programmare noi stessi il controllo quando qualcun altro si è già preso la briga di farlo?

Una libreria deve essere caricata all'inizio di un programma tramite il comando #include.

Il comando per usare la libreria Servo.h:

```
#include <Servo.h>
```

Per utilizzare una libreria, il file deve trovarsi nella cartella di installazione dell'IDE di Arduino. Questo avviene automaticamente all'installazione della libreria tramite IDE.

Nel prossimo capitolo, impareremo come installare l'IDE di Arduino e le funzioni più importanti in modo semplice.

5. L'IDE di Arduino

L'IDE (ambiente di sviluppo integrato) di Arduino è un programma gratuito che, come tutto il sistema Arduino, è progettato per essere facile da usare e dare una struttura al programma.

Con l'aiuto dell'IDE di Arduino, è possibile programmare tutti i modelli di Arduino, Arduino Nano, Arduino Uno o Arduino Mega.

L'installazione dell'IDE di Arduino è quasi autoesplicativa.

Il sito ufficiale di Arduino può essere raggiunto all'indirizzo arduino.cc. Sotto la sezione Software puoi trovare il download del programma che è disponibile per Windows, Mac OS X e Linux.

Dato che la maggior parte dei lettori usa un PC Windows, qui verrà trattata l'installazione in Windows. L'installazione per Mac OS X e Linux è molto simile. Nel frattempo, è disponibile anche un'app Microsoft, che può essere scaricata dal negozio associato.

Poiché non tutti hanno un account e non possono usare il negozio, viene mostrato il download alternativo tramite file exe. Il download può essere effettuato anche tramite la cartella zip, ma richiede qualche click in più.

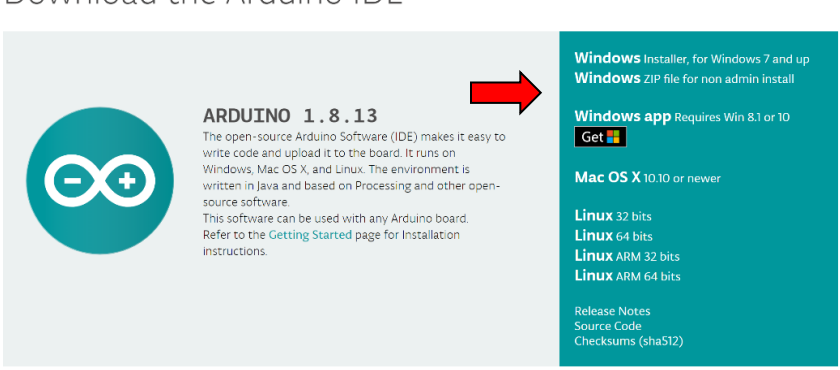


Illustrazione 17 Scaricare l'IDE di Arduino

Prima del download ti apparità una richiesta di donazione. La piattaforma Arduino è completamente gratuita, ed è per questo che si affida alle donazioni.

Tuttavia, se o quanto viene donato non ha alcun effetto sul download o sul programma stesso. Una volta che il download è iniziato, tutto quello che dovrai fare è seguire le istruzioni e installare il software. Il procedimento richiede pochi minuti.

5.1. Struttura di un programma Arduino

Dopo aver installato il software, possiamo aprire l'IDE di Arduino per la prima volta. Ti apparirà uno sketch con data e nome del programma. Lo sketch è il nome del codice del programma ed è formato da

una pagina quasi del tutto bianca, struttura di base di qualsiasi programma Arduino.

Le due sezioni setup e loop sono obbligatorie e devono essere presenti in ogni programma.

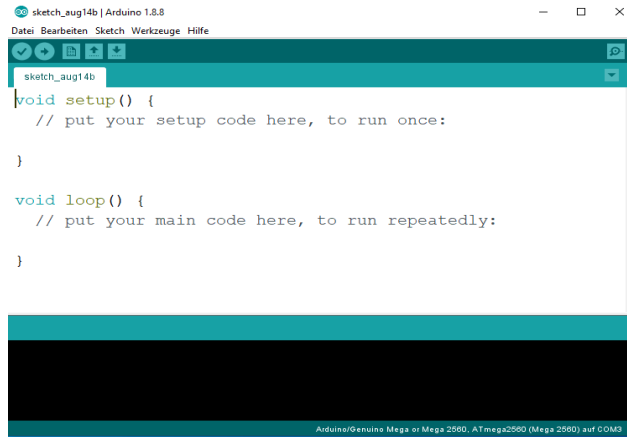


Illustrazione 18 IDE Arduino

5.1.1. void setup(){ }

Void setup(){ } rappresenta una funzione che viene eseguita una volta dopo l'avvio del microcontrollore. Setup è un nome dato di default, ma la funzione può essere rinominata a piacere. Tuttavia, si raccomanda di non cambiare il nome per motivi di chiarezza.

Nella parte di setup(), per esempio, vengono definiti gli ingressi e le uscite o viene inizializzato l'hardware esterno.

Le variabili globali o le librerie, d'altra parte, devono essere definite o incluse al di fuori del setup(), altrimenti non possono essere utilizzate in altre funzioni.

Quando e come qualcosa viene definito o inizializzato sarà discusso di nuovo in dettaglio nel primo esempio pratico (6.1 Controllo dei LED - controllo dei semafori).

5.1.2. void loop(){ }

void loop(){}, invece, descrive un programma che viene eseguito continuamente (loop). Il tipo di dati void è prefissato alle funzioni che non restituiscono un valore.

Dopo l'accensione del microcontrollore, la funzione di setup viene eseguita una volta. Poi la funzione loop viene eseguita e ripetuta all'infinito.

La funzione loop contiene la sequenza effettiva del programma, se un LED deve lampeggiare, i sensori devono essere letti o un motore deve essere mosso.

5.1.3. Aggiungere librerie

Sappiamo già cosa sono le librerie e a cosa servono. Ma qual è il modo più semplice per aggiungerle?

Nell'IDE di Arduino, troverai una panoramica di tutte le librerie, simile a un app store, dove è possibile cercare e installare varie librerie, naturalmente tutte gratuite,. Per farlo, bisognerà andare in Sketch e poi scorrere in basso fino a includi librerie. Tramite qesto comando si apre un menu con tutte le librerie disponibili. Cliccando su Manage Libraries arriviamo a una panoramica dove possiamo aggiornare le nostre librerie attuali. Possiamo anche cercare varie librerie e aggiungerle tramite Install.

Se, invece, scarichiamo una libreria (da Internet) come file zip, questa può essere installata in alternativa tramite Sketch > Include Library > Include Zip Library.

Puoi quindi includere le librerie tramite Sketch > Include Library e selezionando la libreria. In alternativa, potrai includere la libreria manualmente con il comando #include all'inizio di un programma.

5.2. Indent

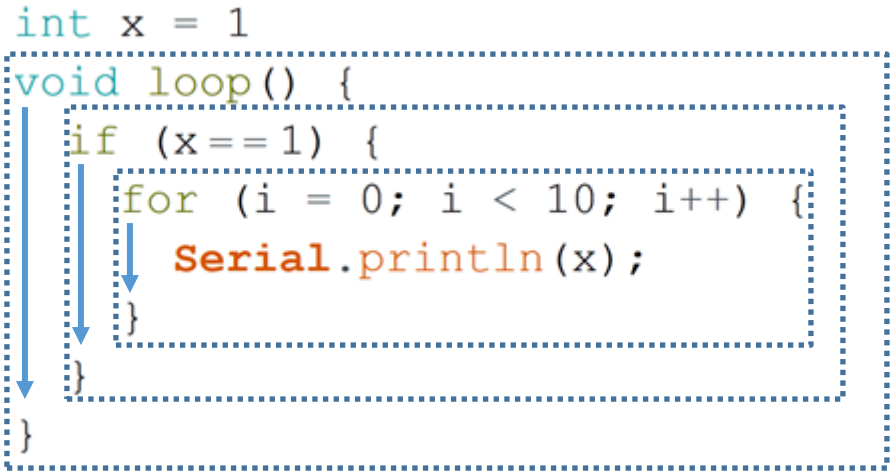
Uno degli strumenti più importanti per rendere chiaro il codice del programma è l'indentazione. Questa può essere usata per strutturare e separare visivamente le parti del programma. I paragrafi possono anche essere usati per una maggior chiarezza all'inizio, ma per progetti più grandi è consigliabile evitare paragrafi più grandi.

```
int x = 1
void loop() {
  if (x==1) {
    for (i = 0; i < 10; i++) {
      Serial.println(x);
    }
  }
}
```

Una regola per il rientro è che ogni nuovo livello di programma viene rientrato.

Un nuovo livello di programmazione inizia dopo le parentesi graffe, come si puuò vedere in una funzione, un ciclo o una dichiarazione.

Il vantaggio è che la parentesi graffa di chiusura '}' è posta direttamente sotto la dichiarazione o il ciclo a cui appartiene. La seguente illustrazione mostra chiaramente i diversi livelli di programmazione. Potrai vedere immediatamente quali linee di codice appartengono all'istruzione if e quali al ciclo for.



Un altro vantaggio: se si segna una parentesi graffa con il mouse, la parentesi corrispondente appare con uno sfondo rosso. Questo ti permette di controllare rapidamente se hai dimenticato una parentesi graffa o se ne hai digitate troppe.

L'IDE di Arduino ha una funzione di auto-formattazione. Per formattare automaticamente il codice basterà cliccare su CTRL+T..

5.3. Funzioni importanti

Come potete immaginare, ci sono ora un numero quasi infinito di progetti, librerie e funzioni. Ogni giorno se ne aggiungono di nuovi.

Di conseguenza, è praticamente impossibile spiegare tutte le funzioni in modo completo. La descrizione di alcune librerie e funzioni è più estesa di questa guida completa per principianti. Pertanto, ci concentreremo solo sulle funzioni più importanti che si trovano in quasi tutti i programmi, per esempio, la funzione di essere in grado di leggere o valutare segnali esterni. Cominciamo prima con le più importanti.

5.3.1. Delay e delaymicrosecondi

Il processore elabora le istruzioni a rotta di collo. Ad un certo punto, però, potrebbe diventare importante consentire al processore di fare una breve pausa nell'esecuzione del programma.

A questo scopo troviamo la funzione delay().

Alla funzione verrà comunicato il tempo desiderato in millisecondi (1000 millisecondi corrispondono a un secondo). Se vogliamo che il programma aspetti due secondi , scriviamo

```
delay(2000);
```

Quando vogliamo aspettare meno di un millisecondo useremo il comando delayMicroseconds(). Vediamo ora come fare in caso di microsecondi. 1000 microsecondi corrispondono a un millisecondo. Se vogliamo aspettare 200 microsecondi, scriviamo

```
delayMicroseconds(200);
```

5.3.2. analogRead()

La funzione analogRead() legge il valore di un pin analogico. Il valore viene restituito alla funzione.

Per esempio, se vogliamo conoscere il valore della voltaggio al pin A5, scriviamo

```
analogRead(A5);
```

Per salvare il valore, dobbiamo poi scriverlo in una variabile.

Poiché Arduino calcola solo in modo digitale, questo valore, che è nell'intervallo da 0 V a 5 V, sarà convertito in un numero.

Le schede Arduino contengono un convertitore analogico-digitale a 10 bit per questo scopo. Il convertitore AD assegna un valore intero tra 0 e 1023 a ciascuna delle tensioni di ingresso sui pin analogici, che sono compresi tra 0 V e 5 V.

Con un Arduino Uno, un cosiddetto stadio di quantizzazione corrisponde quindi a $\frac{5V}{1024} = 0{,}0049\ V = 4{,}9\ mV$

Se volete usare il valore di voltaggio di un pin direttamente come valore nel programma, la conversione è la seguente:

```
Voltaggio in V = analogRead(pin analogico) · 0,0049;
```

La funzione analogRead(); restituisce un valore intero. Se volete memorizzare il valore in una variabile di tipo diverso, deve prima avvenire una conversione di tipo.

5.3.3. map()

La funzione map è spesso usata in combinazione con la funzione analogRead().

La funzione mappa un intervallo di numeri in un altro. È quindi perfetta per convertire il valore che otteniamo dalla funzione analogRead() nella nostra gamma di valori desiderata.

Per fare questo, la funzione ha bisogno del valore da convertire, dei limiti inferiore e superiore dell'intervallo di valori corrente e dei limiti inferiore e superiore del nuovo intervallo di valori.

Per esempio, un valore misurato sul pin analogico A2 deve essere convertito in una voltaggio da 0 V a 5000 mV.

```
int Measured_value = analogRead(A2); // questo valore è nell'intervallo 0...1023

int Voltage_in_mV = map(Measured_value,0,1023,0,5000);
```

Valore misurato -> Valore da assegnare.
0 -> Il limite inferiore della gamma di valori attuali
1023 -> Il limite superiore della gamma di valori correnti.
0 -> Il limite inferiore della nuova gamma di valori.
5000 -> Il limite superiore della nuova gamma di valori.
Il valore viene convertito dal vecchio intervallo di valori 0...1023 al nuovo intervallo di valori 0.... 5000.

Il valore 141 diventa il valore 688, 47convertito in

$$(\frac{141}{1024} \cdot 5000 = 688{,}47)$$

5.3.4. analogWrite - controllo PWM

Simile a come la funzione analogRead() legge i valori analogici, la funzione analogWrite() può emettere un valore analogico. Per fare questo, possiamo usare i pin digitali che sono capaci di PWM. Su Arduino Uno, questi sono i pin 3,5,6,9,10 e 11.

Excursus PWM:

Se avete prestato attenzione, avrete notato che la funzione analogWrite usa i pin digitali per l'uscita.

Come è possibile che un pin digitale che può commutare solo 0 V o 5 V improvvisamente emetta anche valori intermedi come 1 V, 3 V o 4,36 V?

La soluzione è già stata accennata e si chiama Pulse Width Modulation, o PWM in breve che sta ad indicare la modulazione di larghezza d'impulso.

La modulazione di larghezza d'impulso descrive la simulazione di un certo voltaggio (inferiore a 5 V) commutando rapidamente un voltaggio superiore (5 V) seguito da pause senza voltaggio. Arduino accende e spegne l'uscita digitale fino a 1000 volte al secondo. Più alto è il voltaggio di uscita, più lungo è il tempo di accensione durante il quale i 5 V sono presenti.

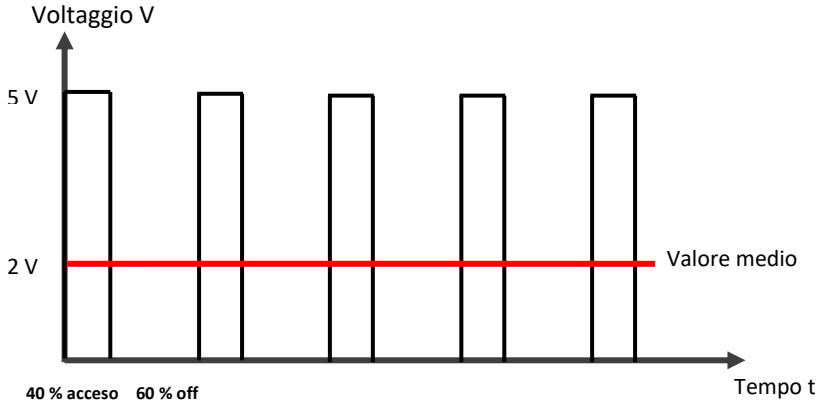


Illustrazione 19 Segnale PWM

Se, per esempio, un LED deve funzionare a 2 V, 5 V viene applicato solo il 40% del tempo. $(\frac{2V}{5V} \cdot 100\%)$. Il restante 60% del tempo è presente 0 V. In media, questo si traduce in 2 V. Poiché il LED si accende e si spegne 1000 volte al secondo, il nostro occhio non percepisce lo sfarfallio, ma una luminosità inferiore. Lo stesso principio funziona anche, per esempio, per il controllo della velocità di un motore DC, come la ventola di un PC.

Come utenti, in realtà non notiamo la modulazione di larghezza d'impulso. Il voltaggio in uscita è praticamente costante a 2 V, a meno che non si misuri con un dispositivo di misurazione molto veloce, come un oscilloscopio.

La funzione analogWrite() ha proprietà simili a analogRead(), ma non restituisce un valorein quanto non viene letto nulla.

Anche qui, il numero del pin PWM corrispondente viene prima passato alla funzione. Inoltre, deve essere trasferito un valore tra 0 e 1023 che corrisponde al voltaggio.

Se vogliamo emettere un voltaggio analogico di 3 V sul pin digitale PWM 9, dobbiamo prima convertire 3 V nel numero intero corrispondente.

$$value = \frac{3V}{5V} \cdot 1023 \approx 614$$

Il comando sarebbe quindi

```
analogWrite(9,614);
```

Le designazioni dei pin si trovano nel capitolo 3.3.1 Capitolo pinout.

5.3.5. digitalRead

digitalRead() legge il valore di un pin digitale. Per fare questo, bisogna dire alla funzione che il pin è usato come ingresso.

Come valore di ritorno otteniamo HIGH, o uno se sono presenti 5 V, oppure LOW, o zero se non c'è voltaggio.

In caso volessimo sapere se il pin 9 (pin digitale) è HIGH o LOW, scriviamo.

```
Int valore;

pinMode(9,INPUT);

valore = digitalRead(9);
```

5.3.6. digitalWrite

Anche con la funzione digitalWrite(), il pin deve prima essere dichiarato come uscita (OUTPUT) e solo poi può essere comunicato alla funzione. Inoltre, la funzione deve sapere se l'uscita deve essere HIGH o LOW.

```
pinMode(9,OUTPUT);

digitalWrite(9,LOW);
```

5.3.7. Serial.print

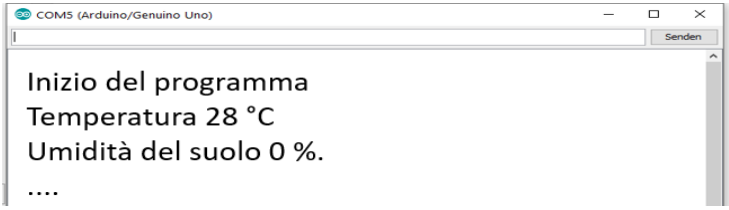
La funzione Serial.print() ci permette di comunicare con Arduino tramite il PC o di leggere la comunicazione. Per fare questo, si passa un valore alla funzione, che può essere un numero, una lettera, una parola o simili . La funzione invia poi il contenuto all'interfaccia seriale (USB).

Se abbiamo collegato un cavo USB al computer, possiamo leggere la comunicazione nell'IDE di Arduino. Per fare questo, apriamo una finestra di dialogo sotto Strumenti > Monitor seriale. Qui possiamo vedere cosa sta succedendo sull'interfaccia seriale, ad esempio, quando Arduino comunica con un altro dispositivo.

Per poter utilizzare l'interfaccia seriale, bisogna prima aprirla tramite il setup() con la funzione Serial.begin(9600).

Il 9600 indica il baud rate con cui Arduino e il PC comunicano.

La funzione Serial.print() è molto utile per visualizzare parole o dati nel programma. In connessione con la funzione Serial.print(), è possibile emettere dati e controllare, per esempio, se sono stati utilizzati i valori corretti.



La funzione Serial.println() è una delle funzioni utilizzate più di frequente. Questa funzione aggiunge un line feed a fine esecuzione nella finestra di output (del monitor seriale),per garantire maggior chiarezza.

Una panoramica delle funzioni menzionate e molte altre informazioni si possono trovare anche sul sito ufficiale di Arduino all'indirizzo:
https://www.arduino.cc/reference/de/

Lì troverete spiegate tutte le funzioni importanti, le operazioni aritmetiche e molto altro.

6. I primi progetti

6.1. Controllo dei LED - controllo dei semafori

Abbiamo sviluppato numerose basi teoriche. adesso siamo pronti a costruire il primo progetto da zero. Anche se non tutto è stato ancora interiorizzato, vogliamo mettere in pratica tutte le conoscenze acquisite. Non preoccuparti se non tutto funziona all'inizio, pensa al motto: "imparare facendo".

Il primo progetto è la costruzione e la programmazione di una versione in miniatura di un circuito semaforico pedonale, un cosiddetto demand light. Il setup consiste in due semafori. In primo luogo, c'è un semaforo pedonale con i colori rosso e verde, così come un interruttore a pulsante per richiedere una fase verde e un semaforo per auto con i colori rosso, giallo e verde.

I semafori per gli automobilisti dovrebbero essere verdi di default, proprio come in strada. Se il pulsante del semaforo pedonale viene premuto, il semaforo passa al giallo e poi al rosso.

In seguito, le luci pedonali passano al verde e rimangono in questo stato per alcuni secondi. Poi i segnali luminosi vengono commutati al contrario: la luce dei pedoni passa al rosso, quella delle auto al rosso-giallo e poi al verde.

6.1.1. Hardware richiesto

Per pianificare il nostro primo setup, abbiamo bisogno, oltre ad Arduino, di alcuni componenti aggiuntivi.

Il voltaggio di alimentazione per Arduino attraverso il cavo USB è sufficiente, ma si potrebbe anche prevedere l'uso di una batteria a blocchi o di un power bank.

Poi, naturalmente, abbiamo bisogno di un Arduino. Come già detto, usiamo un Arduino Uno perché ha abbastanza connessioni, può essere facilmente alimentato e ha delle testate di pin preparate. Abbiamo anche bisogno di vari LED e di alcuni cavi. Per i LED, vengono utilizzati LED da 5 mm con un voltaggio diretto di 3-3,4 V e una corrente di 10 mA. Abbiamo bisogno, inoltre, anche di una resistenza in serie.

Quanto deve essere grande la resistenza in serie dei LED?

Soluzione: Possiamo calcolare facilmente la resistenza in serie dei LED. Nel nostro esempio, dovrebbero cadere un massimo di 2 V (5 V di voltaggio di alimentazione meno 3 V di voltaggio diretto) e circa 10 mA dovrebbero fluire attraverso il LED o la resistenza. Questo risulta in un valore di resistenza di

$$\frac{2V}{10\ mA} = 200\Omega$$

Con un piccolo margine di sicurezza, usiamo 220 Ω che ci consente di avere abbastanza buffer ed è appena percettibile nella luminosità.

A seconda dei LED utilizzati, la stessa resistenza in serie deve essere calcolata.

La seguente tabella mostra l'hardware utilizzato per questo progetto.

Scopo	**Componente**
Microcontrollore	Arduino Uno REV 3
Alimentazione	Cavo USB
2 x luce rossa	5 mm LED rosso
2 x luce verde	5 mm LED verde
1 x luce gialla	5 mm LED giallo
Pulsante	Pulsante a 4 pin
Resistenza serie LED	220 Ω
Resistenza di pull-down	10 kΩ

Invece di LED individuali, ci sono anche moduli semaforici già pronti che constano degli stessi LED saldati semplicemente su un circuito con le loro connessioni.

Per azionare i semafori usiamo un pulsante con quattro perni. Questo pulsante è poco costoso ed è incluso in molti starter kit.

Il pulsante in pressione collega i contatti diagonalmente opposti. Quindi, quando uno di questi contatti è collegato a 5 V e il contatto opposto si trova al pin 2 dell'ingresso digitale di Arduino, si applicano5 V al pin numero 2, corrispondente a un HIGH logico.

Abbiamo anche bisogno di una resistenza pull-down.

Excursus sulle resistenze pull-up e pull-down:

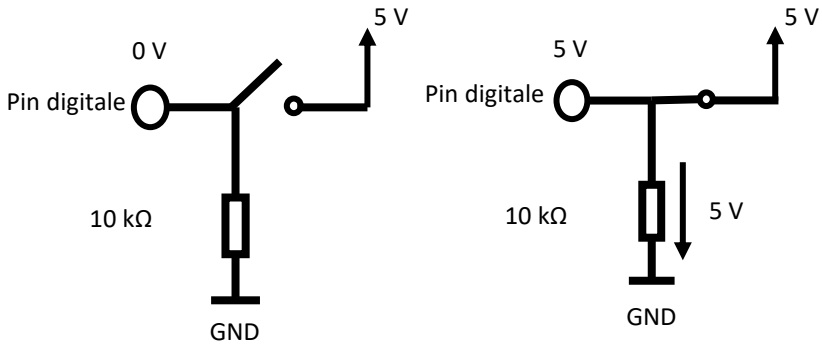
Quando si usano segnali digitali, si usa spesso il termine resistenza pull-up e pull-down. Si tratta di una resistenza ad alta impedenza (di solito 4,7 kΩ o 10 kΩ).

Lo scopo di una tale resistenza è di assicurare che un segnale definito sia sempre presente sul pin digitale. Nel nostro esempio, questo significa che quando il pulsante non è premuto, vogliamo che al pin digitale siano presenti 0 V. Tuttavia, non c'è alcun collegamento con GND. Il pin digitale non è automaticamente a 0 V, ma è "sospeso in aria".

Pertanto, colleghiamo una resistenza tra GND e il pin. Poiché nessuna corrente fluisce in un pin d'ingresso, non cade nessuna voltaggio sul resistore. Questo significa che anche il pin avrà un voltaggio di 0 V che verrà letto come uno 0 logico. Se invece il pulsante è chiuso, la resistenza impedisce un collegamento da 5 V a 0 V.

Con una resistenza di pull-up, la resistenza si trova tra il pin digitale e il voltaggio di alimentazione. Con un resistore pull-down, il resistore viene posizionato tra il pin digitale e la terra (GND).

Pulsante aperto	Pulsante chiuso



6.1.2. Schema del circuito

Il passo successivo è quello di impostare l'hardware per il progetto. L'illustrazione mostra come collegare i vari componenti.

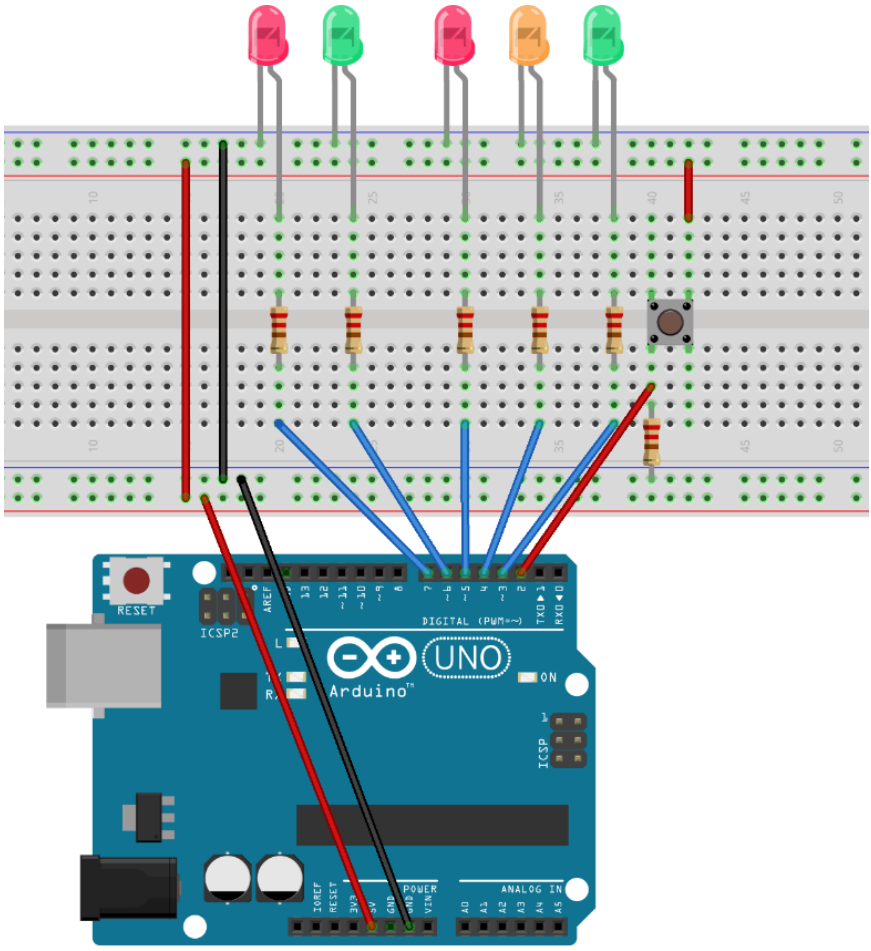


Illustrazione 20 Schema del circuito

Usiamo l'uscita a 5 V di Arduino come alimentazione per i LED e non il pin Vin. Questo è collegato, per esempio, al voltaggio della batteria da 9 V. Qualora decidessimo di colelgare quest'ultimo, dovremmo calcolare una resistenza in serie più grande per i LED, altrimenti i LED potrebero bruciari..

Dopo aver selezionato l'hardware e impostato lo schema del circuito, costruiamo il circuito su una breadboard.

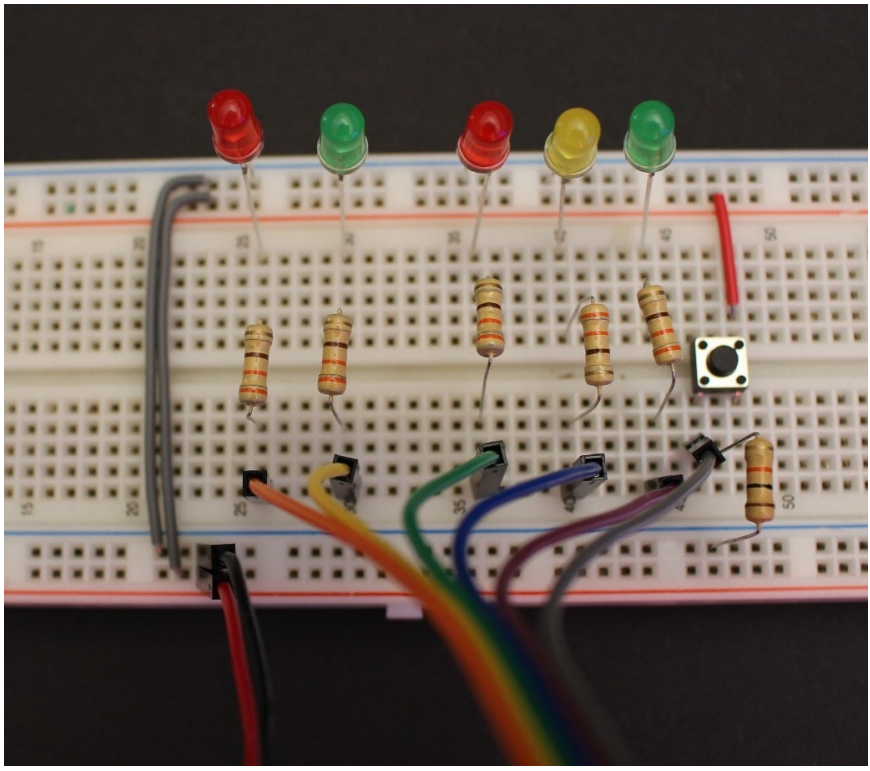


Illustrazione 21 Montaggio sulla scheda a innesto

I cavi portano direttamente agli ingressi e alle uscite di Arduino.

6.1.3. Programmazione

L'ultima cosa che facciamo è la programmazione. Per fare questo, colleghiamo Arduino al PC e avviamo l'IDE di Arduino. Il codice per generare la luce pedonale è relativamente semplice. L'intestazione con autore e data è stata omessa per semplicità.

```
//////Circuito semaforico pedonale V1.0///////

int tempo_attesa = 1000 ; // Tempo tra i processi di commutazione

int tempo_verde = 3000; // fase verde pedonale

const int PIN_rosso_pedoni = 7; // Pin del LED rosso pedonale

const int PIN_verde_pedoni = 6; // Pin del LED verde pedonale
```

```
const int PIN_rosso_semaforo = 5; // Pin del LED rosso Auto

const int PIN_giallo_semaforo = 4;

const int PIN_verde_semaforo = 3;

const int PIN_switch = 2;
```

In primo luogo, vengono create delle costanti per i pin. Non è più necessario scrivere direttamente il pin in questa fase, questo passaggio potrà essere effettuato in seguito. Tuttavia, vediamo rapidamente che definire le costanti una volta sola ha senso.Per esempio, se si fanno dei cambiamenti, si deve modificare solo un valore, non tutti i valori delle variabili o delle costanti nell'intero programma.

Il tempo di inattività è il tempo che passa tra un cambio di colore e l'altro. Inoltre, c'è una fase verde, cioè il tempo durante il quale la luce pedonale è verde. Nell'esempio, il tempo di inattività è di un secondo e la fase verde è di tre secondi.

```
void setup()

{

 //Definire come ingressi e uscite

 pinMode( PIN_rosso_pedoni, OUTPUT);

 pinMode( PIN_verde_pedoni , OUTPUT);

 pinMode( PIN_rosso_semaforo , OUTPUT);

 pinMode( PIN_giallo_semaforo , OUTPUT);

 pinMode( PIN_verde_semaforo , OUTPUT);

 pinMode( PIN_switch , INPUT);


 //condizione di partenza semaforo pedonale rosso - semaforo auto verde

 digitalWrite( PIN_rosso_pedoni, HIGH );

 digitalWrite( PIN_verde_pedoni, LOW );

 digitalWrite( PIN_rosso_semaforo, LOW );

 digitalWrite( PIN_giallo_semaforo, LOW );
```

```
 digitalWrite( PIN_verde_semaforo, HIGH );
```

}Dopo aver impostato le costanti, viene eseguita la parte di setup dove vengono impostati anche gli stati di partenza. Nel nostro esempio, il semaforo delle auto deve essere verde e il semaforo dei pedoni deve essere rosso.

```
void loop() {
if (digitalRead(PIN_switch) ) { // solo quando l'interruttore è premuto
   delay(tempo_attesa); // aspetta un secondo
   digitalWrite( PIN_verde_semaforo, LOW );
   digitalWrite( PIN_giallo_semaforo, HIGH );
   delay(tempo_attesa);
   digitalWrite( PIN_giallo_semaforo, LOW );
   digitalWrite( PIN_rosso_semaforo, HIGH );
   delay(tempo_attesa);
   digitalWrite( PIN_rosso_pedoni, LOW);
   digitalWrite( PIN_verde_pedoni, HIGH);
   delay(tempo_verde);
   digitalWrite( PIN_verde_pedoni, LOW);
   digitalWrite( PIN_rosso_pedoni, HIGH);
   delay(tempo_attesa);
   digitalWrite( PIN_giallo_semaforo, HIGH );
   delay(tempo_attesa);
   digitalWrite( PIN_rosso_semaforo, LOW );
   digitalWrite( PIN_giallo_semaforo, LOW );
   digitalWrite( PIN_verde_semaforo, HIGH );
  }
}
```

La funzione loop controlla se l'interruttore è premuto.

Se questo è il caso, la condizione if (digitalRead(PIN_switch)) è vera e l'istruzione if tra parentesi graffe viene eseguita.

In questo caso, il semaforo delle auto è impostato prima sul giallo e poi, dopo un secondo, sul rosso e quindi il semaforo pedonale sul verde. Dopo la fase verde, il semaforo pedonale diventa rosso e il semaforo automobilistico diventa prima rosso-giallo e poi di nuovo verde. Si consiglia di leggere le istruzioni passo dopo passo e seguirle.

6.1.4. Carica del codice

Dopo che il programma è stato scritto, il passo successivo è quello di caricare il codice nella memoria di Arduino. Per fare questo, Arduino è collegato al computer o al portatile tramite il cavo USB. Arduino viene riconosciuto e qualsiasi driver mancante viene installato automaticamente. Con alcune repliche dall'Estremo Oriente, può succedere che si debbano installare dei driver speciali.

Dopo che Arduino è stato collegato, selezioniamo sotto

Strumenti -> Scheda -> "Arduino/Genuino Uno".

Inoltre, sotto

Strumenti -> Porta -> "COM5 (Arduino/Genuino Uno)".

Il numero della porta COM può essere diverso e dipende, per esempio, da quanti altri dispositivi sono collegati al computer.

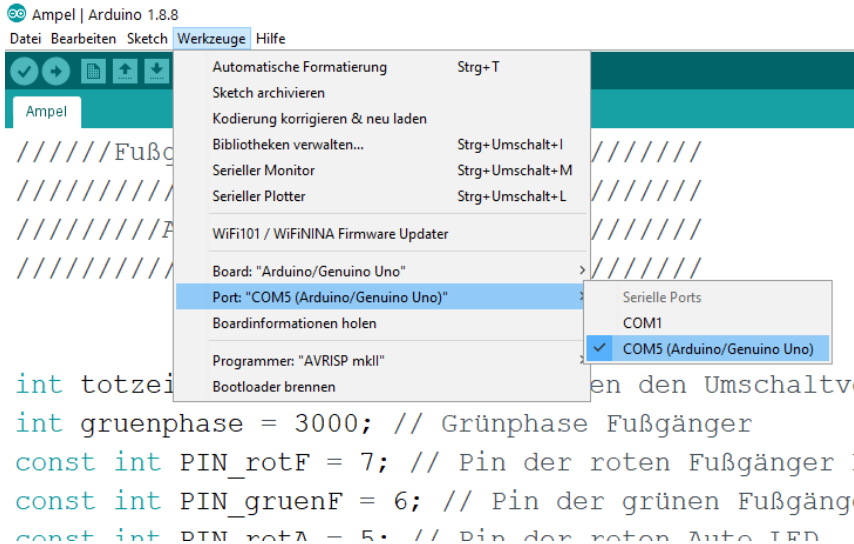


Illustrazione 22 Selezione della porta COM

Dopo aver selezionato la scheda e la porta corretta, il programma può essere caricato tramite la freccia.

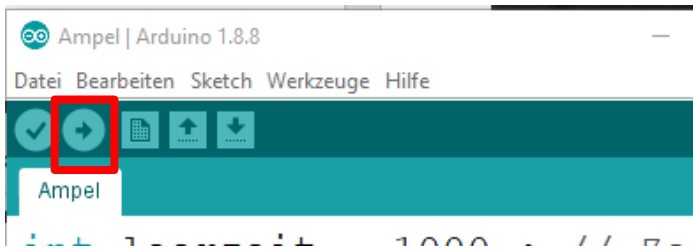


Illustrazione 23 Caricamento del codice del programma

Infine, otteniamo un riassunto di quanta memoria è necessaria per il programma. Nel nostro piccolo esempio solo

4 % della memoria disponibile.

Dopo che il programma è stato caricato su Arduino, il LED verde del semaforo auto e il LED rosso del semaforo pedonale si accendono continuamente. Quando si preme il pulsante, il semaforo si accende come abbiamo programmato prima.

Se l'ordine non corrisponde, controlla che i cavi siano collegati ai pin corretti di Arduino.

Una panoramica del progetto finito:

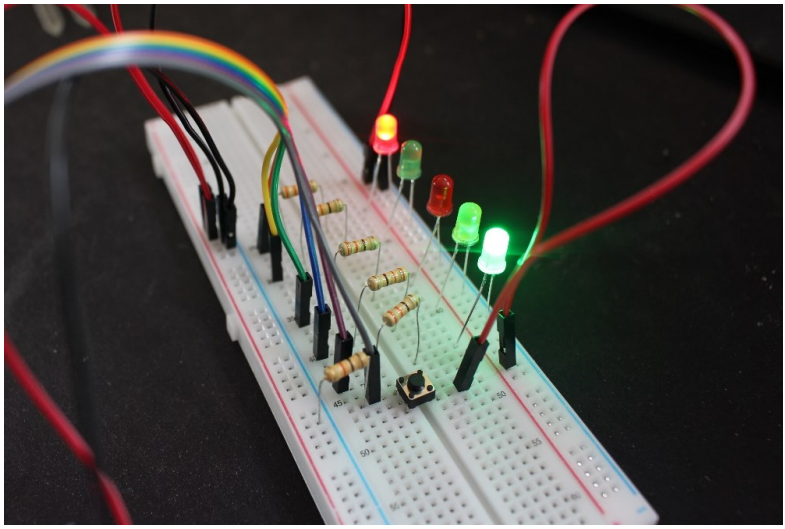


Illustrazione 24 Struttura del progetto finito

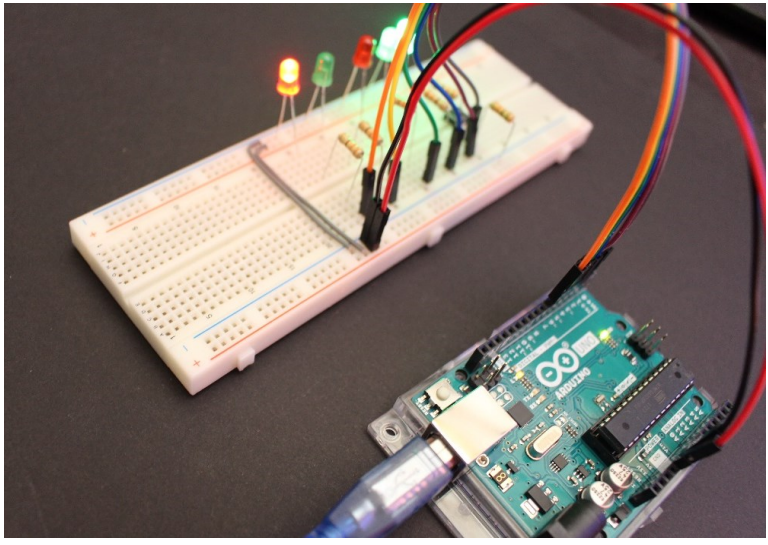


Illustrazione 25 Struttura del progetto finito

6.2. Controllo del servomotore tramite potenziometro

Ora che abbiamo imparato a controllare i LED, questo esempio si occupa del controllo dei motori. Distinguiamo tra diverse classi di motori. Ci sono motori che richiedono solo una voltaggio di alimentazione, ad esempio i motori a corrente continua, e ci sono motori che richiedono hardware aggiuntivo per il controllo, ad esempio i motori passo-passo o i motori a corrente continua senza spazzole.

Tutti questi motori possono essere controllati e la loro velocità regolata tramite un Arduino. In questo esempio usiamo un tipo di motore che viene usato molto spesso in combinazione con Arduino - il servomotore.

Un servomotore può determinare la posizione del suo albero e può quindi essere regolato con molta precisione. Ecco perché i servomotori sono ideali per i progetti di robotica..

A differenza dei motori passo-passo, che possono anche essere regolati, i servomotori sono spesso più economici e non richiedono hardware aggiuntivo.

Il servomotore è controllato tramite Arduino, la posizione angolare del suo albero è determinata tramite un potenziometro rotativo.

6.2.1. Hardware richiesto

Come sempre, abbiamo bisogno di un'alimentazione. Nel nostro progetto controlleremo solo un servomotore, quindi l'alimentazione tramite USB e il convertitore di voltaggio interno a 5 V di Arduino è sufficiente. Se in un progetto vengono utilizzati diversi servomotori, è essenziale collegare un'alimentazione esterna.

Scopo	Componente
Microcontrollore	Arduino Uno REV 3
Alimentazione	Cavo USB

Attuatore	Servomotore MG90S
Specifica della posizione	B100potenziometro

Servomotore

Un servomotore è un motore DC che ha un sensore incorporato. Questo determina la posizione angolare del suo albero motore. Inoltre, il servomotore ha un riduttore per aumentare la forza o la coppia del motore e ridurre la velocità di rotazione.

Per il nostro progetto usiamo un motore MG90S. Questo è quasi identico nella costruzione al motore un po' più economico SG90.

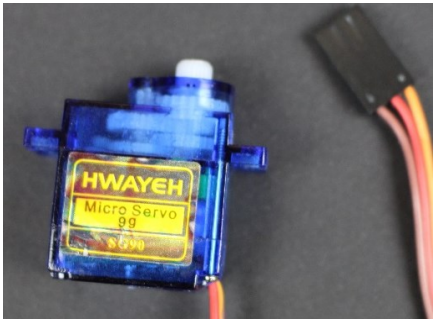


Illustrazione 26 Servomotore MG90S *Illustrazione 27 Servomotore SG90*

L'abbreviazione MG nella designazione del servomotore sta per 'metal geared'. Ciò significa che il riduttore del servomotore è composto da ingranaggi in metallo e non in plastica, più soggetti all'usura, il che consente un tempo di funzionamento notevolmente più lungo.

La posizione dei sensori può essere regolata da 0° a 180°. Il motore è controllato tramite un segnale modulato a larghezza d'impulso (segnale PWM, vedi 5.3.4 analogWrite - controllo PWM)

Inoltre, i motori sono alimentati con 4,8 V - 6 V. Il consumo di corrente a 4,8 V è di 100 mA. Tuttavia, un servomotore può anche assorbire molta più corrente per un breve periodo.

Il servomotore ha due pin per la voltaggio di alimentazione (Vcc e GND) e un pin di segnale che è collegato direttamente a un'uscita digitale di Arduino.

6.2.2. Programmazione

Teoricamente, potremmo calcolare noi stessi quale segnale PWM appartiene a quale posizione del servo. Poi dovremmo generare un segnale PWM e inviarlo al servomotore.

Come spesso accade, ci sono librerie già pronte che fanno proprio questo.

Pertanto, usiamo la libreria "Servo.h".

```
#include <Servo.h>
```

Dopo aver incluso la libreria, dobbiamo dare un nome al nostro motore o dichiararlo come oggetto nel programma.

Questo viene fatto prima della parte di conIllustrazionezione con

```
Servo servo1; // Crea oggetto
```

Un pin di uscita viene quindi assegnato al nostro servomotore nell'illustrazione. Nel nostro esempio, questo è il pin di uscita digitale 2, che non ha bisogno di essere dichiarato ulteriormente come pin di uscita.

```
servo1.attach(2); // pin digitale 2 - segnale PWM per il servo
```

Infine, possiamo specificare la posizione del servomotore con la funzione servo1.write(Position). La posizione deve essere compresa tra 0° e 180°. servo1.write(80), per esempio, imposta il servo a 80°.

Potenziometro

Un potenziometro, è una resistenza rotante. Internamente, consiste in due resistenze regolabili collegate in serie. Un contatto scorrevole aumenta la resistenza superiore o quella inferiore, e l'altra resistenza diminuisce di conseguenza.

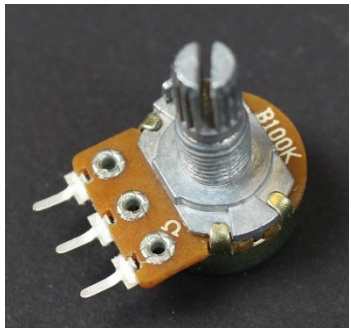
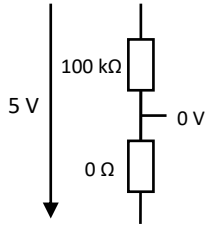
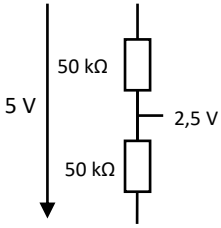
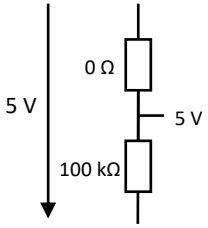


Illustrazione 28 Potenziometro

Il potenziometro ha tre spine di collegamento. C'è una resistenza permanente di 100 kΩ tra le spine esterne. La resistenza tra il centro e i contatti esterni dipende dalla posizione della ruota.

Se colleghiamo il nostro voltaggio di alimentazione di 5 V ai contatti esterni, otteniamo un divisore di voltaggio regolabile.

Così girando il potenziometro possiamo influenzare il rapporto dei due valori di resistenza e quindi la voltaggio al pin centrale.

Posizione inferiore	Posizione centrale	Posizione superiore
		

6.2.3. Schema del circuito

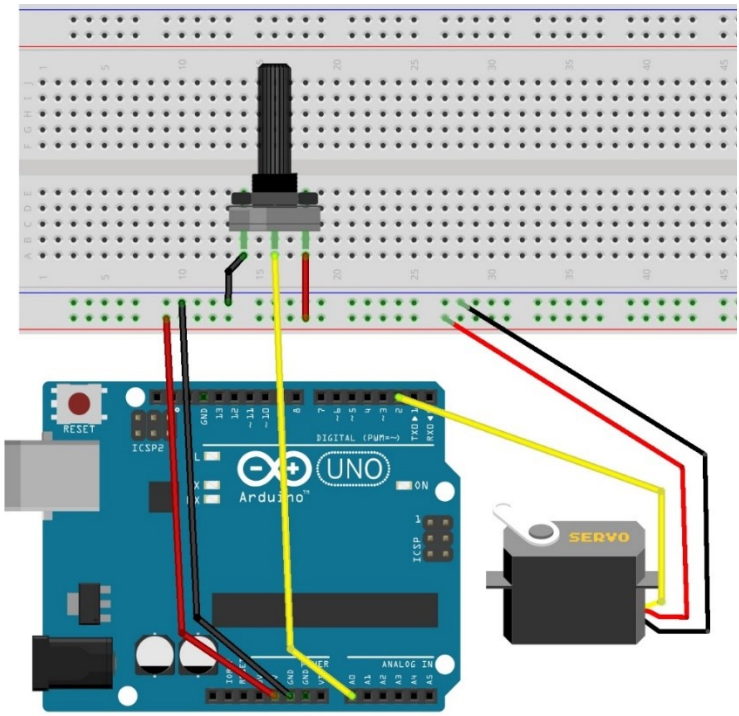


Figura 29 Schema del circuito

Lo schema del circuito è molto semplice. Il servomotore e il potenziometro sono collegati all'alimentazione di Arduino. Il pin di segnale del servo è collegato al pin digitale 2 e il punto medio del potenziometro al pin analogico A0.

6.2.4. Programmazione

Anche la programmazione è molto semplice. Prima si integra la libreria e si inizializza il servomotore, poi si definisce una variabile in cui viene memorizzato il voltaggio del potenziometro, così come destinato all'uscita della posizione di grado del servo.

Nella parte obbligatoria di setup, il pin di uscita 2 è assegnato al servomotore. Il pin PWM del servomotore è collegato al pin 2.

```
#include <Servo.h>
```

```
Servo servo1;                    // Crea oggetto
int pot;                  // Variabile in cui viene letta la voltaggio
int serv;                  // Variabile per l'uscita del grado
void setup() {
 servo1.attach(2);      // pin digitale 2 - segnale PWM per il servo
}
```

Nella funzione loop, il valore di voltaggio viene letto e riscalato in un numero di gradi da 0 a 180. La posizione viene quindi emessa.

```
void loop() {
 pot = analogRead(A0); // legge il valore analogico (0... 1023)
 serv = map(pot, 0, 1023, 0, 180); // scala a 180
 servo1.write(serv);
 delay(100);
}
```

Un ritardo di 100ms alla fine del ciclo dà al servomotore abbastanza tempo per assumere la posizione prima che ne venga specificata una nuova.

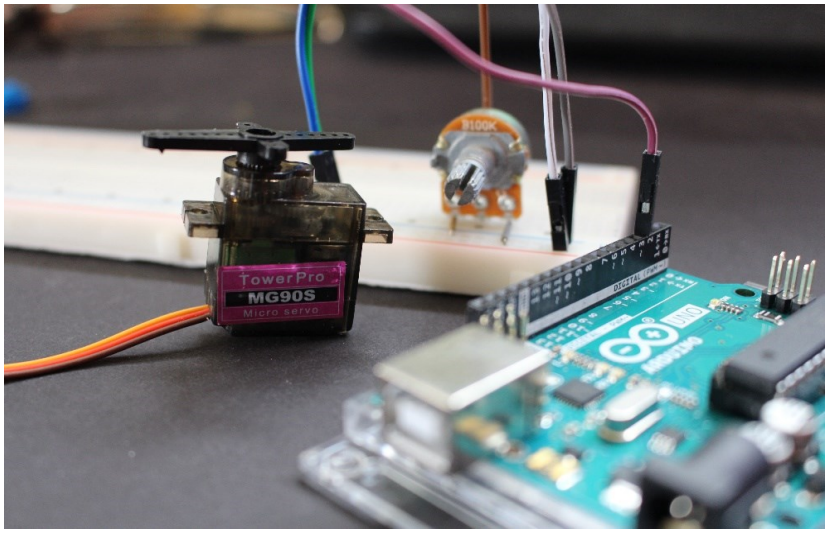


Illustrazione 30 Struttura del progetto finito

Dopo che il codice è stato caricato su Arduino, si può girare il potenziometro. Anche il servomotore reagisce girando.

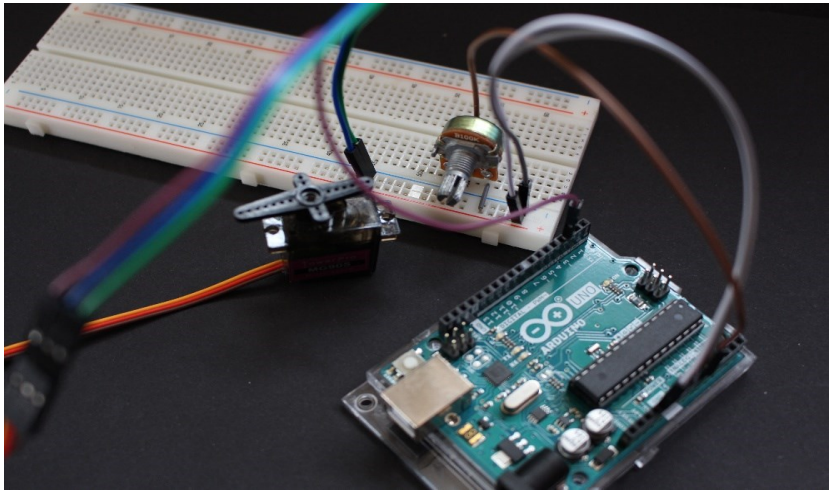


Illustrazione 31 Struttura del progetto finito

Sembra che il potenziometro e il servomotore siano collegati direttamente tra loro, ma il controllo è realizzato elettronicamente attraverso l'elaborazione dei dati in Arduino.

6.3. Sensori di controllo - Stazione meteorologica

Dopo aver costruito e programmato i nostri primi piccoli progetti, arriviamo ora a un progetto più ampio.

La piattaforma Arduino offre molti sensori, alcuni dei quali spieghiamo qui a titolo di esempio. A questo scopo, combiniamo il controllo di diversi sensori e un display nella struttura dell'esercizio di questo capitolo.

Questo esempio riguarda l'impostazione e la programmazione di una stazione meteorologica.

Lo scopo della stazione meteorologica è quello di misurare la temperatura, l'umidità, la luminosità e l'umidità del suolo, mostrando tutto su un display. Nel processo, impariamo ad usare diversi sensori.

6.3.1. Hardware richiesto

La stazione meteorologica deve prima essere alimentata con energia elettrica. Per questo abbiamo bisogno di una voltaggio di alimentazione, per esempio un alimentatore da 9 V. Si potrebbero usare anche un power bank o una batteria a blocchi. Per questo esempio, usiamo una batteria a blocco da 9 V in modo che la stazione possa essere utilizzata anche senza fili.

Per prima cosa, ovviamente, abbiamo bisogno di un Arduino. Come già detto, usiamo un Arduino Uno perché ha abbastanza connessioni, può essere facilmente alimentato e ha dei pin header. Se si vuole ridimensionare il progetto, si raccomanda invece un Arduino Nano.

Abbiamo anche bisogno dei vari sensori, così come di alcuni cavi e della nostra scheda plug-in. Infine, ma non meno importante, visualizziamo i dati di misurazione su un display OLED.

La seguente tabella mostra l'hardware utilizzato:

Scopo	Componente

Microcontrollore	Arduino Uno REV 3
Alimentazione	Cavo USB
Temperatura e umidità	DHT22
Umidità del suolo	Resistenza all'umidità
Luminosità	Fotoresistenza
Rilevamento della pioggia	Sensore di pioggia
Visualizza	Display OLED 128x64
Resistenza di pull-down	10 kΩ

Sensore di temperatura e umidità

Ci sono molti modi diversi per determinare la temperatura. Una variante molto economica è una resistenza PTC o NTC. Un PTC (coefficiente di temperatura positivo) è un resistore il cui valore di resistenza aumenta con la temperatura.

Se conosciamo la curva caratteristica del resistore in funzione della temperatura, possiamo misurare il valore della resistenza per ricavare la temperatura. Il vantaggio di questo metodo di misurazione è che un PTC standard può essere ottenuto per pochi centesimi. Lo svantaggio, d'altra parte, è che devi costruire tu stesso un circuito per misurarlo. Dopo tutto, Arduino non può leggere valori di resistenza, solo valori di voltaggio. Quindi, per costruire un divisore di voltaggio abbiamo bisogno di un altro resistore. Il voltaggio che cade sulla PTC è proporzionale al valore della resistenza della PTC.

Un metodo semplice ma anche più costoso per misurare la temperatura sono i moduli intelligenti già pronti. Misurano la temperatura, la digitalizzano direttamente e inviano i dati ad Arduino tramite un'interfaccia.

Un esempio di tale dispositivo è il DHT22. Questo è un sensore combinato di temperatura e umidità. Misura la temperatura dell'aria e l'umidità relativa e trasmette i valori su richiesta tramite l'interfaccia seriale. Questo sensore è ideale per il nostro progetto. Con esso, abbiamo registrato sia l'umidità che la temperatura in un solo componente.

Il DHT22 è il "fratello maggiore" del DHT11. Sono in gran parte identici, solo la precisione del DHT22 è significativamente migliore. Il DHT22 ha un'impercettibile imprecisione di 0,2°, mentre il DHT11 ha un'imprecisione fino a 2 °C.

Anche il cablaggio è molto semplice. Il sensore ha quattro pin.

In alternativa, ci sono già schede di circuiti che portano fuori solo i tre pin necessari. Un pin rimane inutilizzato e non ha alcuna funzione.

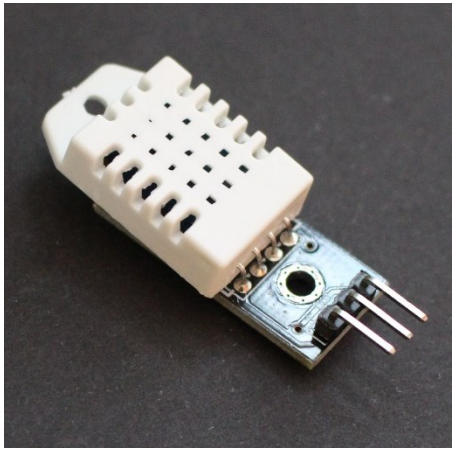
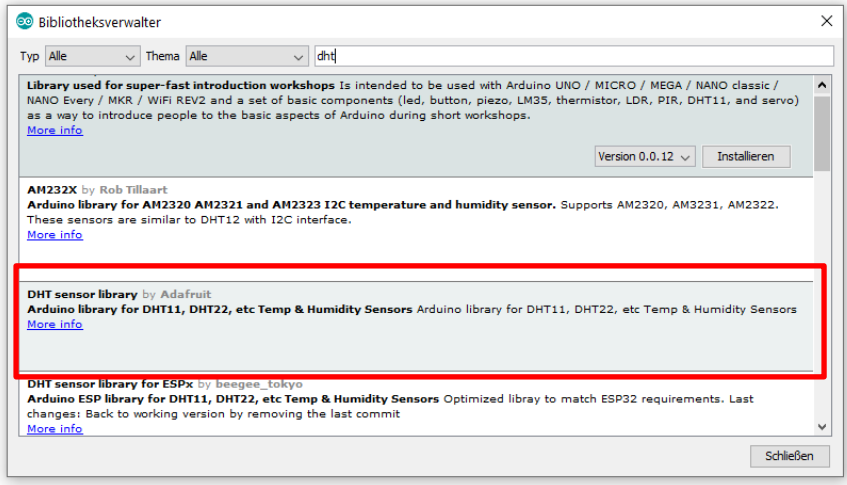


Immagine 32 Sensore DHT22

Il sensore nell'illustazione richiede una voltaggio di alimentazione e un pin GND. C'è anche un pin da collegare a un pin di ingresso digitale di Arduino.

Programmazione del sensore di umidità

Per la programmazione, scarichiamo prima la libreria DHT. Come spiegato nel capitolo 5.1.3 cerchiamo la libreria tramite Strumenti -> Gestisci librerie



Dopo che la libreria è stata installata, possiamo usarla nel codice del programma con

```
#include "DHT.h"
```

La libreria è fatta su misura per il DHT11 e il DHT22 e offre semplici funzioni per leggere l'umidità e la temperatura.

Prima, però, dobbiamo definire a quale pin digitale è collegato il sensore e quale sensore è. Nel nostro esempio, usiamo un DHT22 sul pin digitale 2 di Arduino.

```
DHT dht(2, DHT22); //DigitalPin 2 e tipo DHT22
```

Con i comandi

dht.readTemperature() e dht.readHumidity() possiamo quindi leggere facilmente la temperatura e l'umidità. Nel programma appare così:

```
float temp; // Temperatura in °C con decimale

int umidita_aira; // Umidita relativa in %.

temp = dht.readTemperature();

umidita_aira = dht.readHumidity();
```

Umidità del suolo

Usiamo un sensore appositamente progettato per misurare l'umidità del suolo. Il sensore si chiama FC-28 e il principio di misurazione è semplice.

Si tratta di due piastre metalliche che vengono semplicemente inserite nel terreno e che hanno una resistenza alta o bassa l'una con l'altra a seconda del valore di umidità.

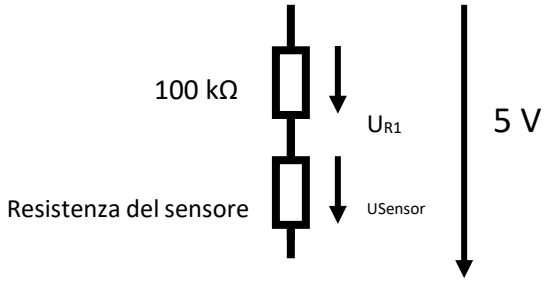
L'acqua conduce l'elettricità, quindi sappiamo che l'umidità del suolo dipende dal valore della resistenza.

Il sensore di umidità del suolo è un resistore la cui conducibilità è determinata e dipendente dall'umidità del suolo.

Per ottenere un valore di voltaggio che possiamo successivamente valutare, dobbiamo costruire un divisore di voltaggio (2.5.1).

Esempio di un divisore di voltaggio

Colleghiamo il sensore e una resistenza da 100 kΩ in serie e picchiettiamo nel punto medio. Quindi misuriamo la voltaggio attraverso il resistore



Per il voltaggio attraverso un divisore di voltaggio vale quanto segue

$$\frac{voltaggio\ alla\ resistenza}{voltaggio\ totale} = \frac{valore\ della\ resistenza}{resistenza\ totale}$$

$$Usensor = \frac{resistenza\ del\ sensore}{resistenza\ del\ sensore + 100k\Omega} \cdot 5V$$

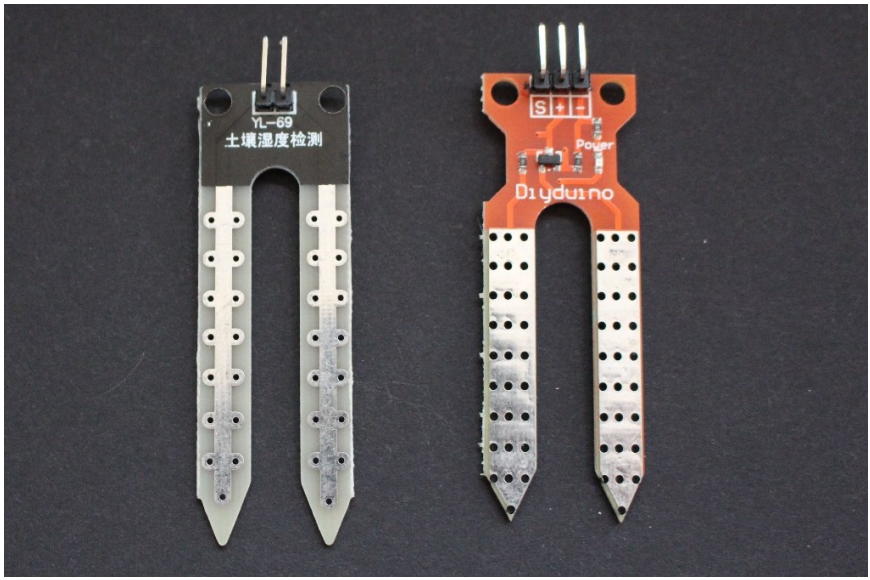


Illustrazione 33 Sensore di umidità del suolo senza (sinistra) e con divisore di voltaggio (destra)

In alternativa, lo stesso sensore è disponibile con un circuito integrato in cui è già incorporato un divisore di voltaggio adatto (illustrazione a destra). Questi sensori "pronti all'uso" sono raccomandati per i principianti. Il sensore ha un pin Vcc (voltaggio di alimentazione), un pin GND e un pin di uscita analogica.

Il vantaggio del sensore utilizzato è che converte il valore di resistenza direttamente in un voltaggio da 0 V (corrisponde allo 0 % di umidità del suolo) a un voltaggio di alimentazione di 5 V (corrisponde a un suolo senza resistenza, quindi conduttivo).

Tuttavia, il divisore di voltaggio è utilizzato per lo schema del circuito in modo che si possa costruire il circuito anche senza un tale modulo già pronto. La differenza è solo un resistore in più sulla scheda del plug-in. Il valore della resistenza dipende dal modulo utilizzato ed è dell'ordine di 10-100 kΩ.

Programmazione del sensore di umidità del suolo:

Nel nostro modulo, 0 V corrisponde a un'umidità del suolo dello 0 % e 5 V corrisponde a un'umidità del suolo del 100 %. Letto in analogico, questo significa che 0 % di umidità del suolo corrisponde al valore 0 e 100 % di umidità del suolo al valore 1023.

A seconda del sensore, è possibile che l'intera gamma di valori non venga utilizzata. Con il sensore utilizzato, la gamma di valori non raggiunge un massimo di 1023 ma solo 878.

Con un divisore di voltaggio autocostruito, dovete prima calcolare il campo di voltaggio e quindi il campo di valori.

Con il comando map() in connessione con analogRead(), c'è solo un comando con cinque parametri per la lettura.

```
int umidita_suolo; // Umidita del suolo in %.

const int PIN_umidita_suolo = A0; // ingresso analogico 0

umidita_suolo = map(analogRead(PIN_umidita_suolo), 0, 878, 0, 100);

 //converte il valore del sensore (0...878) in percentuale (0..100)
```

Il valore letto dal pin analogico A0 è assegnato alla variabile Bmoisture (umidità del suolo). Questo valore (0...878) viene prima convertito in un valore percentuale (0...100). Per maggiori informazioni sulla funzione map(), vedere in 5.2.3.

Sensore di luce

Per poter misurare la luminosità, usiamo una fotoresistenza. Questa resistenza cambia il suo valore a seconda dell'illuminamento. Al buio, il fotoresistore ha un valore di resistenza di circa 20 kΩ. Se è completamente illuminato, per esempio con una torcia, il valore di resistenza si riduce a meno di 100 Ω.

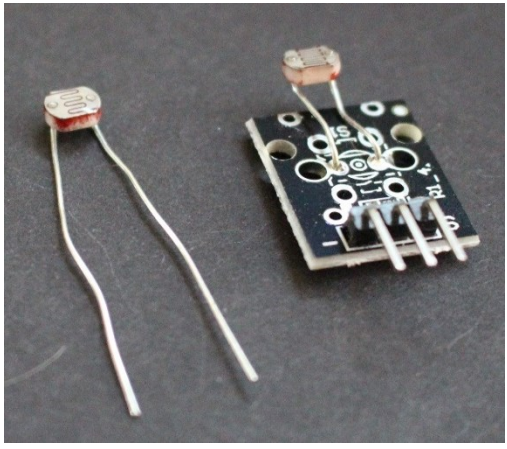


Figura 34 Resistenza leggera senza (sinistra) e con divisore di voltaggio (destra)

Analogamente al sensore di umidità del suolo appena discusso, che è anche una resistenza variabile, anche qui si deve costruire un divisore di voltaggio.

Naturalmente, ci sono anche moduli di sensori già pronti in questo settore. Bisogna notare che la resistenza diminuisce quando c'è molta luce, quindi il segnale deve ancora essere invertito.

Programmazione del sensore di luce:

La programmazione corrisponde esattamente a quella del sensore di umidità del suolo, solo la gamma di valori è stata adattata.

```
int luminosita; // luminosita in %

const int PIN_luminosita = A1; // ingresso analogico 1

luminosita = map(analogRead(PIN_luminosita), 0, 1023, 100, 0);

//converte il valore del sensore (0...1023) in percentuale (0..100)
```

Sensore di pioggia

Il sensore di pioggia è il tipo di sensore più semplice nella nostra disposizione di esempio. Usiamo un cosiddetto sensore di pioggia FC-83 (identico nella costruzione al sensore YL-83).

Si tratta solo di due contatti di binari conduttori. Questi non hanno un collegamento, ma possono essere cortocircuitati da una goccia d'acqua. Se i contatti sono collegati, c'è acqua sul sensore. Se i binari non sono collegati, non c'è acqua sul sensore.

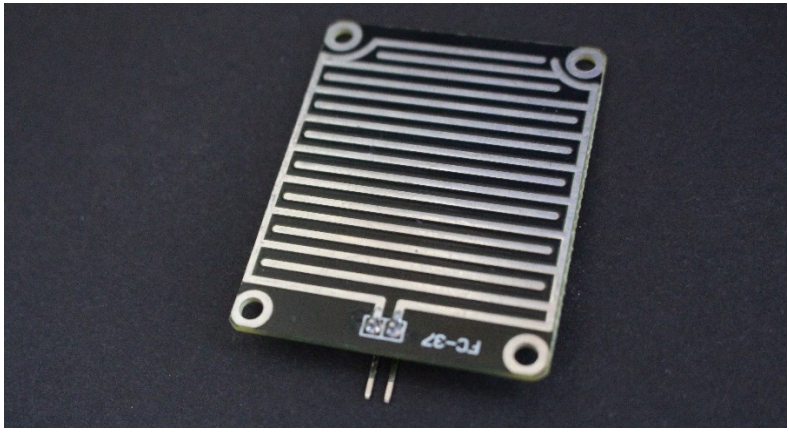


Illustrazione 35Sensore di pioggia CF-37

Di conseguenza, applichiamo 5 V a un pin del sensore e portiamo l'altra estremità a un pin di ingresso digitale. Se c'è anche 5 V su questo pin, sappiamo che c'è una connessione, cioè che sta piovendo.

Abbiamo anche bisogno di una resistenza pull-down collegata tra il pin digitale e GND. Perché il sensore di pioggia non è altro che un interruttore.

Programmazione del sensore di pioggia:

Il pin digitale viene interrogato. Se c'è una connessione, è presente un HIGH e la parola "sì" è memorizzata nella variabile rain. Se non c'è pioggia, "no" viene memorizzato di conseguenza.

```
const int PIN_pioggia = 3; // pin digitale 3

String pioggia; // Pioggia sì o no

if (digitalRead(PIN_pioggia)) { // quando piove il pin è ALTO

  pioggia = "sì";

 }

 else {

  pioggia = "no";

 }
```

Visualizza

I valori che i sensori trasmettono al controller sono mostrati su un display OLED da 0,96 pollici con 128x64 pixel.

Il display comunica con il microcontrollore tramite il bus I²C. Abbiamo bisogno di due linee per il bus I²C. Una linea è usata come orologio per specificare quando i dati sono trasmessi. Si chiama SCL (clock seriale). L'altra linea è usata per inviare i dati effettivi avanti e indietro. Questa linea ha l'abbreviazione SDA (dati seriali).

L'esatta modellazione e valutazione dei dati sul bus I²C è meno importante per noi. Ci concentriamo sui controlli attraverso le librerie esistenti. Per fare questo, dobbiamo prima collegare il display con due pin per il voltaggio di alimentazione Vcc e GND, così come i due pin dati SCL e SDA descritti sopra.

Diversi display OLED di diversi produttori hanno diverse assegnazioni di pin. Vcc e GND sono talvolta scambiati. Assicurarsi sempre che il cablaggio sia corretto!

I pin per il voltaggio di alimentazione sono collegati ai 5 V di Arduino e alla terra tramite la scheda di connessione.

Per la connessione seriale via I²C, Arduino ha i pin SDA e SCL, a sinistra dei pin digitali I-/O.

Programmazione del display OLED

Ci sono molte librerie diverse per i display I²C. Di seguito, viene utilizzata la libreria "U8glib.h". Contiene semplici comandi per visualizzare lettere latine, numeri arabi e caratteri speciali. In primo luogo, la libreria viene installata tramite il gestore delle librerie.

Successivamente, viene riempito con

```
#include "U8glib.h"                    //libreria per la visualizzazione
```

caricato nell'Arduino.

Poi, dobbiamo dire al microcontrollore che abbiamo collegato un display. Creiamo un oggetto con il comando

```
U8GLIB_SH1106_128X64 oled(U8G_I2C_OPT_NONE);
```

Al nostro display è stato dato il nome di "oled". "U8G_I2C_OPT_NONE" sono le opzioni I²C standard, che non approfondiremo.

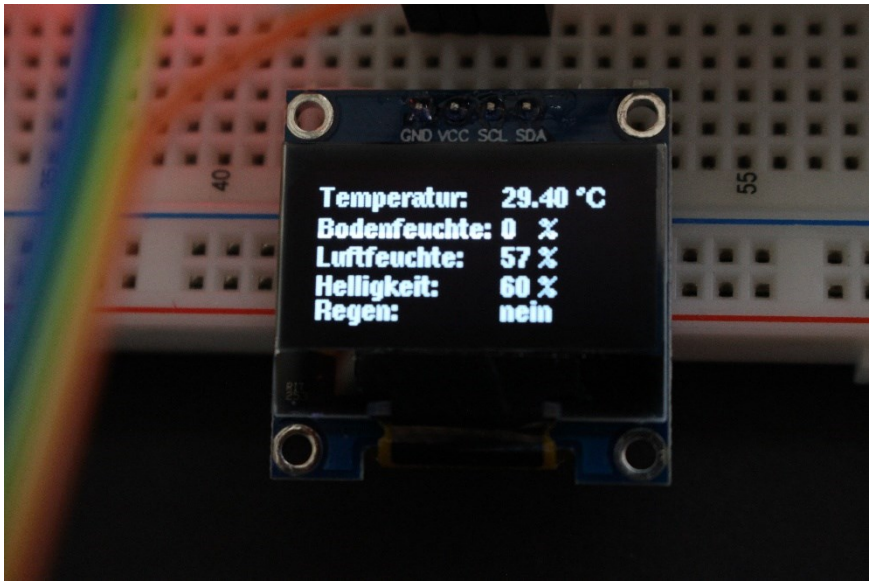


Illustrazione 36 Uscita dati sul display OLED

Il passo successivo è quello di visualizzare le diverse variabili e parole. Le seguenti funzioni sono a nostra disposizione per questo scopo:

```
oled.firstPage(); Crea la "prima pagina" del display dopo l'inizio del programma.
```

oled.nextPage(); Indica se può essere creata una "pagina successiva". Le due funzioni sono usate in combinazione per creare un ciclo che aggiorna costantemente il display e sovrascrive i dati precedenti.

```
oled.setFont(); Qui potete impostare il tipo di testo, la dimensione del carattere, ecc.

oled.print(); stampa una parola o una variabile sul display

oled.write(); Lo stesso di oled.print per i caratteri speciali
```

oled.setPrintPos(X, Y); Imposta un cursore virtuale nella posizione (X|Y) (coordinate in pixel). Il carattere successivo viene stampato in questa posizione.

Dopo aver conosciuto le funzioni più importanti della libreria U8glib, iniziamo a programmare.

Per questo creiamo una funzione Display_Output. Non appena questa funzione viene eseguita, i dati vengono rinnovati.

Ecco un esempio di codice per la visualizzazione della prima riga

```
void Display_Output() {
 oled.setFont(u8g_font_helvB08);              //dimensione carattere 8, carattere * grassetto
 oled.setPrintPos(0, 15);            // Imposta il cusore su X=0, Y=15 pixel
 oled.print("Temperatura: ");
 oled.setPrintPos(75, 15);           // Imposta il cusore su X=75, Y=15 pixel
 oled.print(temp);      // restituisce il contenuto della variabile temp
 oled.setPrintPos(105, 15);
 oled.write(0xb0);      // Codice per il segno °
 oled.print("C");
```

Ripetiamo la stessa procedura per tutte e quattro le linee.

Nel programma principale (loop loop) chiamiamo la funzione.

Il ciclo do-while esegue la funzione finché la funzione oled.nextPage(); non restituisce un errore. Questo accade solo in caso di errore.

```
oled.firstPage(); //crea la prima pagina
 do { //run....
   Display_Output();
 } while ( oled.nextPage() ); //... a condizione che la pagina successiva sia "vera"
}
```

6.3.2. Schema del circuito

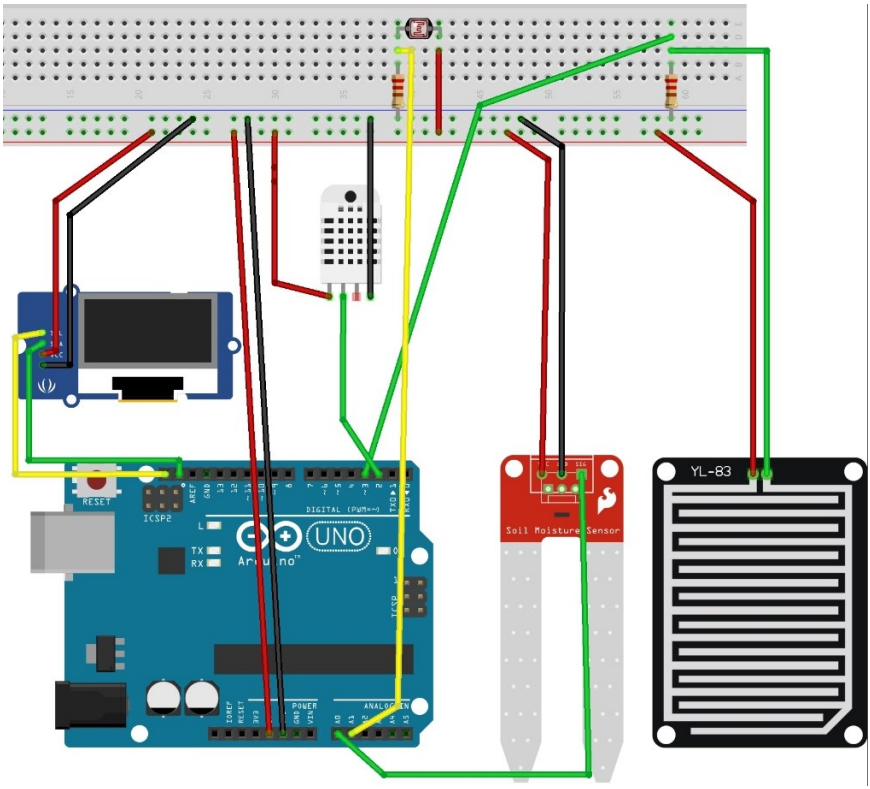


Illustrazione 37 Schema del circuito

Tutti i moduli sono forniti con un voltaggio di alimentazione di 5V.

Il pin digitale del modulo DHT22 è collegato al pin digitale 2 di Arduino.

Il sensore di pioggia è collegato al pin digitale 3 di Arduino tramite una resistenza pull-down da 10 kΩ.

Il sensore di umidità del suolo è collegato al pin di ingresso analogico A0.

Un divisore di voltaggio è impostato per la fotoresistenza. Se si utilizza un modulo con divisore di voltaggio integrato, questo resistore viene omesso. Il segnale intercettato viene alimentato nel pin analogico A1.

6.3.3. Tabella dei collegamenti

DHT22	Pin Arduino
Vcc (+)	5 V
GND (-)	GND
SIG (Gio)	Pin 2

Misuratore di umidità del suolo FC-28	Pin Arduino
Vcc (+)	5 V
GND (-)	GND
SIG (Gio)	A0

Sensore di pioggia FC-37	Pin Arduino
Pin 1 (+)	5 V
Pin 2 (-)	Pin 3

Modulo fotosensore	Pin Arduino
Vcc (+)	5 V
GND (-)	GND
SIG (Gio)	A1

Display OLED	Pin Arduino
Vcc (+)	5 V
GND (-)	GND
SCL	SCL
SDA	SDA

6.3.4. Programmazione

Abbiamo già suddiviso la programmazione dei singoli sensori. Di seguito, combineremo le singole parti in un programma coerente. Cominciamo sempre con l'includere le biblioteche. Creiamo anche un oggetto per il display e per il sensore DHT22. Poi impostiamo tutte le variabili e le costanti.

```
//////stazione meteorologica///////

#include "DHT.h"

#include "U8glib.h" //libreria per la visualizzazione

U8GLIB_SH1106_128X64 oled(U8G_I2C_OPT_NONE); //crea l'oggetto display

DHT dht(2, DHT22); //DigitalPin 2 e tipo DHT22



float temp; // temperatura in °C

int umidita_aira; // Umidita relativa in %.

int umidita_suolo; // Umidita del suolo in %.

int luminosita; // luminosita in %

String pioggia; // Pioggia sì o no



const int PIN_pioggia = 3; // pin digitale 3

const int PIN_umidita_suolo = A0; // ingresso analogico 0

const int PIN_luminosita = A1; // ingresso analogico 1
```

Prima di eseguire il setup(), viene scritta la funzione completa Display_Output().

```
void Display_Output() {

 oled.setFont(u8g_font_helvB08); //font di dimensione 8 e font spesso

 oled.setPrintPos(0, 15); // Imposta il cusore su X=0, Y=15 pixel

 oled.print("Temperatura: ");

 oled.setPrintPos(75, 15); // Imposta il cusore su X=75, Y=15 pixel

 oled.print(temp); // restituisce il contenuto della variabile temp

 oled.setPrintPos(105, 15);

 oled.write(0xb0); // Codice per il segno °

 oled.print("C");

 // Uscita dell'umidita del suolo nella linea 2

 oled.setPrintPos(0, 28);
```

```
  oled.print("Umidita del suolo:");
  oled.setPrintPos(75, 28);
  oled.print(umidita_suolo);
  oled.setPrintPos(90, 28);
  oled.print("%");
  // Uscita dell'umidita dell'aria nella linea 3
  oled.setPrintPos(0, 40);
  oled.print("Umidita:");
  oled.setPrintPos(75, 40);
  oled.print(umidita_aira);
  oled.setPrintPos(90, 40);
  oled.print("%");
  // Uscita della luminosita nella linea 4
  oled.setPrintPos(0, 52);
  oled.print("Luminosita:");
  oled.setPrintPos(75, 52);
  oled.print(luminosita);
  oled.setPrintPos(90, 52);
  oled.print("%");
  // Gli stati della pioggia in uscita nella linea 5
  oled.setPrintPos(0, 62);
  oled.print("Pioggia:");
  oled.setPrintPos(75, 62);
  oled.print(pioggia);
}
```

La parte del setup è mantenuta molto piccola. Qui, solo l'ingresso digitale per il sensore di pioggia è definito come INPUT e la comunicazione con il DHT22 è iniziata con il comando dht.begin().

```
void setup() {

 dht.begin();

 pinMode( PIN_pioggia, INPUT);

}
```

Nella funzione principale, tutti i dati del sensore vengono letti e convertiti. Poi viene eseguita la funzione Display_Output() per visualizzare i dati sul display OLED.

```
void loop() {

 if (digitalRead(PIN_pioggia)) { // quando piove il pin è ALTO

   pioggia = "sì";

 }

 else {

   pioggia = "no";

 }

 umidita_suolo = map(analogRead(PIN_umidita_suolo), 0, 878, 0, 100); //converte il valore del sensore (0...878) in percentuale (0..100)

 luminosita = map(analogRead(PIN_luminosita), 0, 1023, 100, 0); //converte il valore del sensore (0...1023) in percentuale (0..100)

 temp = dht.readTemperature();

 umidita_aira = dht.readHumidity();



 oled.firstPage(); //crea la prima pagina

 do { //run....

   Display_Output();

 } while ( oled.nextPage() ); //... a condizione che la pagina successiva sia "vera"
```

```
  delay(60000); // Un minuto di ritardo

}
```

Poiché una misura deve essere presa una volta al minuto, un "ritardo" di 60.000 millisecondi (60 secondi) alla fine del ciclo fornisce il ritardo corrispondente.

Teoricamente, dovremmo ancora sottrarre il tempo che il microcontrollore ha bisogno per eseguire gli altri comandi. Ma è solo una frazione di secondo.

Il programma viene caricato di nuovo tramite la freccia in alto a sinistra.

Dopo che il programma è stato caricato con successo, il display OLED mostra i valori di temperatura, umidità del suolo, umidità e luminosità, nonché lo stato della pioggia. Una volta al minuto, i valori misurati vengono riletti e visualizzati sul display.

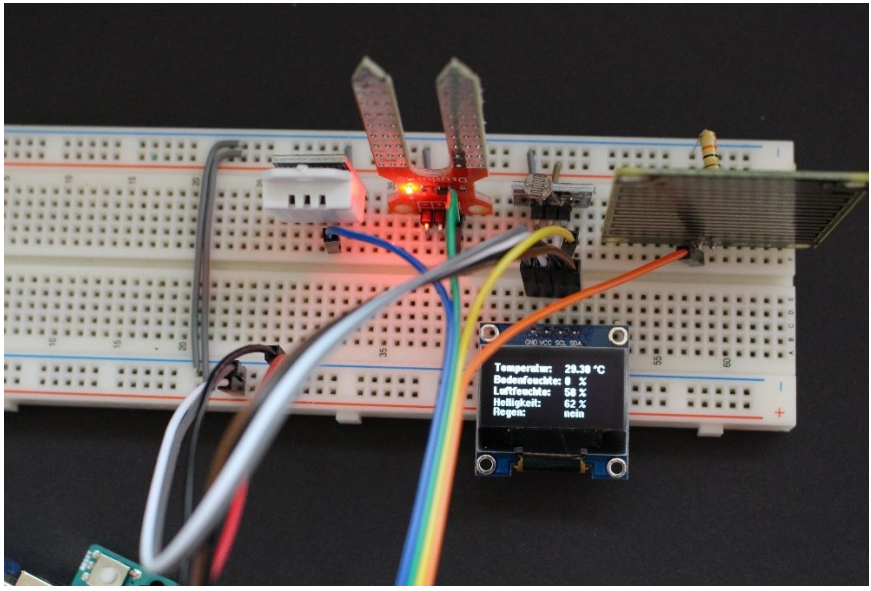


Illustrazione 38 Struttura del progetto finito

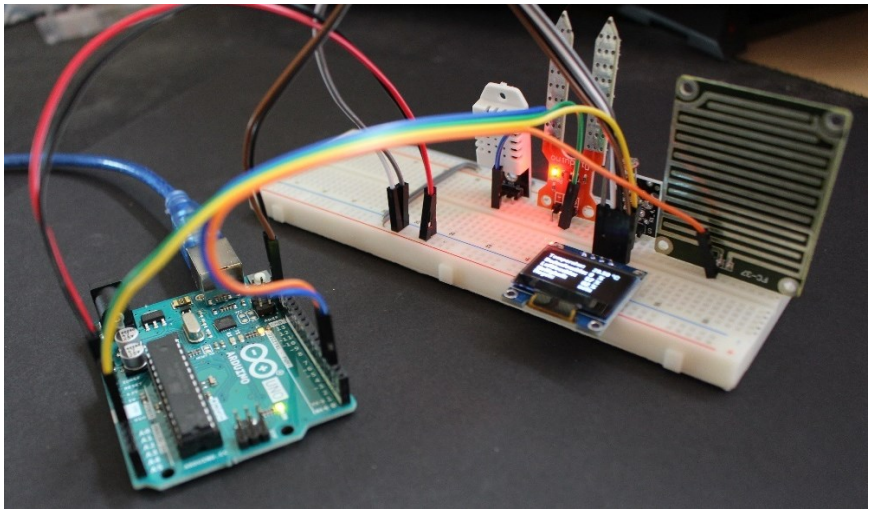


Illustrazione 39 Struttura del progetto finito

7. Gli errori più comuni dei principianti

Nessun maestro è mai caduto dal cielo. Tutti commettono errori, specialmente quando sono principianti in una materia. Il vantaggio di una guida come questa, tuttavia, è che possiamo imparare dagli errori degli altri senza doverli fare noi stessi. Ecco perché passiamo in rassegna gli errori più comuni sia in termini di software che di hardware.

7.1. Cortocircuiti, fumo, calore

Se il dispoitivo comincia a diventare caldo o addirittura a fumare, l'alimentazione deve essere rimossa immediatamente. Di solito si tratta di un cortocircuito, per esempio perché i pin dell'alimentazione sono stati invertiti (Vin o 5 V e GND). Controllare tutte le connessioni per vedere se c'è un corto circuito da qualche parte. Spesso può succedere anche perché avete collegato due file sulla breadboard in modo non previsto. A 3.7 possiamo vedere lo schema di collegamento della scheda a a innesto.

Una resistenza in serie troppo piccola per un LED o un transistor può anche causare una possibile fonte di calore. Se tutti i cavi sono stati controllati per i cortocircuiti, un componente difettoso può anche essere la fonte dello sviluppo di calore. Prima di farlo, però, è consigliabile analizzare attentamente il cablaggio e lo schema del circuito per vedere se c'è un errore di progettazione, in modo da non "sacrificare" altri componenti.

7.2. Il controller si blocca - i componenti non funzionano

Se il controller Arduino si blocca, di solito è perché l'alimentazione sta cedendo. È opportuno, pertanto, controllarla attentamente.. Verifica che l'alimentatore sia di buona qualità e che possa fornire abbastanza potenza. Se Arduino è alimentato da un power bank o da una batteria a blocchi, controlla se il livello di carica è sufficiente.

Quando ci si connette via USB, il voltaggio può avere una caduta di tensione se la porta USB è troppo debole e il cavo è troppo lungo.

D'altra parte, può essere un problema di carico che influisce sul voltaggio e lo fa cadere. Per esempio, se si usano dei motori, questi possono assorbire una

quantità relativamente grande di corrente quando si avviano, e l'alimentazione si guasterà di conseguenza.

Come soluzione, si raccomanda di sostituire l'alimentatore con uno di alta qualità. Per i progetti con molta carica (ad esempio i motori), si dovrebbe sempre collegare una fonte di voltaggio supplementare. Per progetti maggiori si utilizza uno scudo Arduino con prese di ingresso per un alimentatore esterno.

Se i componenti non funzionano correttamente o si guastano completamente, può essere che siano alimentati con troppo o troppo poco voltaggio. Per capire bene il tipo di voltaggio, ti aiuterà dare un'occhiata alle specifiche. Per lo più, i componenti vengono fatti funzionare con 3,3 V o 5 V. Inoltre, se l'alimentazione è superiore a 5 V, bisogna fare attenzione che il pin Vin non sia usato per alimentare i componenti, poiché è collegato direttamente alla presa d'ingresso.

7.3. La comunicazione bus non funziona

Per esempio, si usa un display I²C, ma la comunicazione non funziona. Il bus I²C ha bisogno di due resistenze di pull-up per SDA e SCL. Di solito questi sono integrati direttamente sui componenti (per esempio il display). Nel caso di linee lunghe, diversi consumatori sul bus o alimentazione instabile, può aiutare a commutare due resistenze da 4,7 kΩ di SDA e SCL contro 5 V.

7.4. Il led non si accende

Se un LED dovrebbe accendersi ma non lo fa, può essere un errore hardware o software. Il LED è probabilmente installato nel modo sbagliato (polo positivo sulla gamba più lunga) o la resistenza in serie è stata scelta troppo grande. È importante considerare quanto voltaggio di flusso e quanta corrente richiede il LED.

D'altra parte, se la resistenza in serie è troppo bassa, il LED può surriscaldarsi e bruciarsi.

Se nessuna di queste possibilità si applica, c'è un errore del software. L'OUTPUT digitale potrebbe essere stato dichiarato come INPUT per errore, potrebbe essere stato specificato il pin sbagliato o un errore di programma potrebbe impedire l'attivazione del pin digitale.

7.5. Nessun file o directory di questo tipo

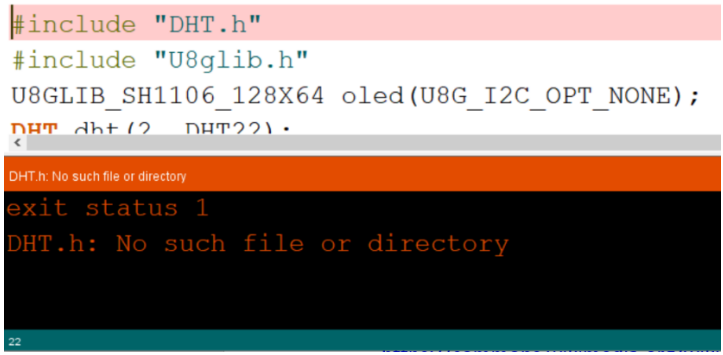


Illustrazione 40 Errore - nessun file o directory di questo tipo

In questo errore, manca la libreria da includere. Nell'esempio, la libreria DHT.h non è disponibile e deve essere prima installata. Vedere 5.1.3 Aggiungere librerie.

7.6. "Atteso ';' prima di ' ... ' gettone"

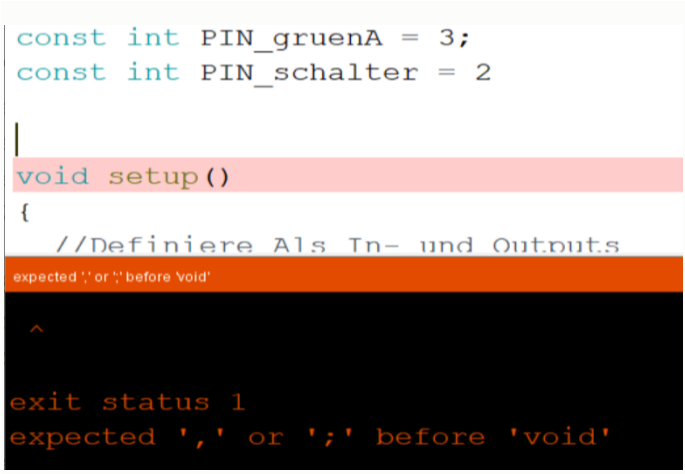


Illustrazione 41 Errore-atteso ',' o ';' prima di 'void'.

Questo errore indica che è stato dimenticato un punto e virgola dopo una dichiarazione. Di solito è appena sopra la linea segnata in rosso. Nell'esempio, il punto e virgola dopo

```
const int PIN_switch = 2;
```

7.7. "Atteso inizializzatore prima di ' ... ' "

Questo errore, o spesso aspettarsi `}' alla fine dell'input, è altrettanto comune quando si è nuovi alla programmazione.

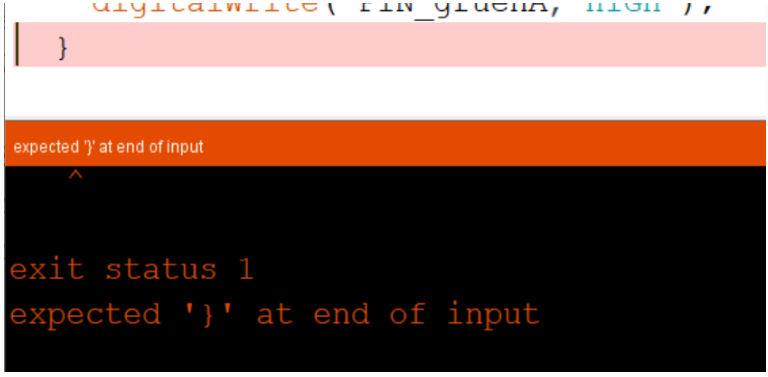


Illustrazione 42 Errore - ci si aspettava '}' alla fine dell'input

Se appare uno di questi errori, una parentesi graffa è stata dimenticata da qualche parte. Ogni parentesi graffa aperta è sempre accompagnata da una parentesi graffa chiusa.

7.8. "Dichiarazione attesa prima di ' ... ' token

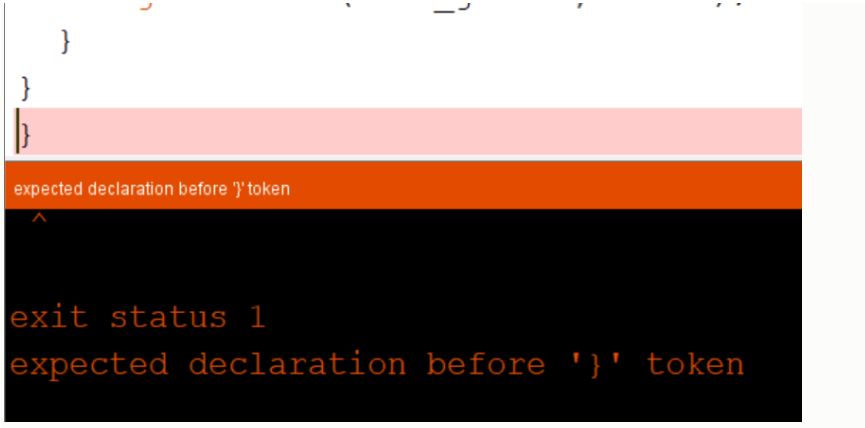


Illustrazione 43 Errore - attesa dichiarazione prima di '}'

Questo messaggio di errore, invece, appare se sono state impostate troppe parentesi graffe.

Anche qui, la formattazione aiuta a controllare. All'inizio, la sostituzione corretta della staffa è noiosa, ma ci si abitua col tempo.

Vedi anche l'aiuto alla formattazione nel paragrafo 5.2 Indentazione.

7.9. "Ridefinizione di ' ... ' "

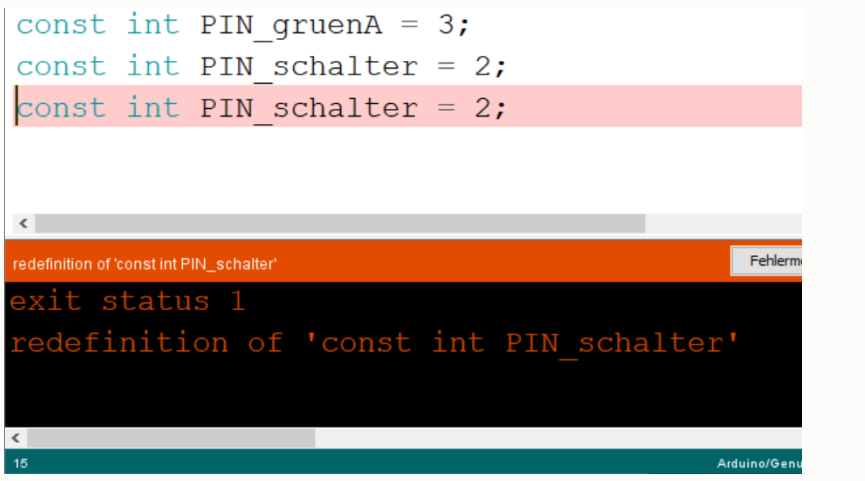


Illustrazione 44 Errore - ridefinizione di '...'.

Qualcosa è stato definito più volte, anche se può verificarsi solo una volta. Questo errore può accadere facilmente quando si aggiungono linee di codice estranee a un programma con il copia e incolla.

7.10. "Problema di caricamento sulla scheda"

Spesso ci sono problemi a trovare la scheda sul PC o a caricare lo sketch. Per prima cosa bisogna vedere se il PC riconosce Arduino.

Per fare questo, apriamo Device Manager sotto Windows e vediamo se Arduino è riconosciuto sotto le porte COM e LPT.

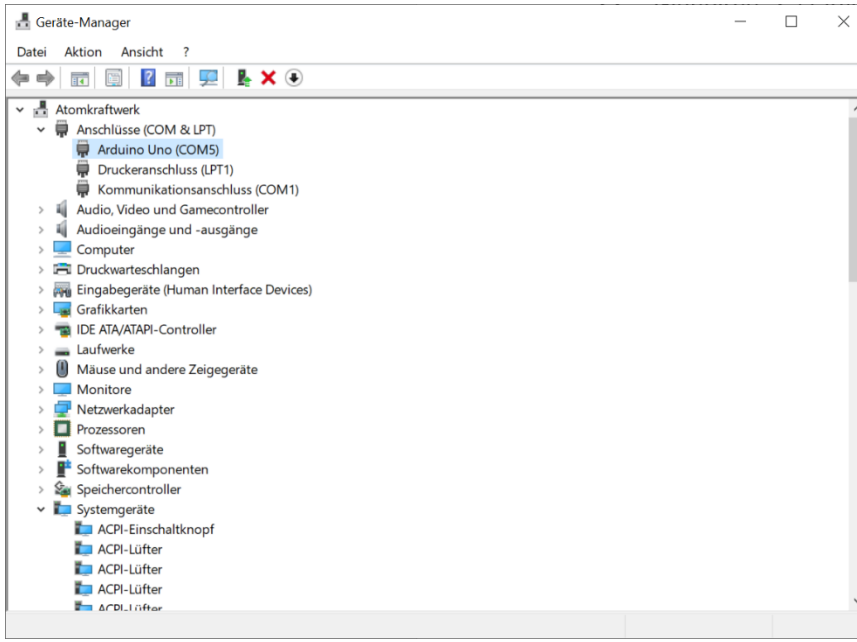


Illustrazione 45 Gestione dispositivi di Windows

Se Arduino viene riconosciuto, è un buon segno. Se una porta COM viene riconosciuta quando Arduino è collegato, ma non è designata come Arduino, molto probabilmente mancano i driver appropriati. Questo succede spesso con i cloni di Arduino. In questo caso, i driver possono essere trovati sui siti web dei produttori.

Se non viene rilevata alcuna porta COM, potrebbe esserci un difetto hardware. Prima, però, si raccomanda di cambiare il cavo USB; spesso i cavi USB non sono

sufficientemente schermati o non sono di qualità sufficiente. Cambiare il cavo USB può già risolvere l'errore. Se si usa un clone di Arduino, potrebbe essere necessario installare prima dei driver adatti.

Dopo che Arduino è stato riconosciuto, bisogna selezionare la porta e la scheda corretta nell'IDE di Arduino.

Se Arduino viene riconosciuto e può essere selezionato, ma lo sketch non può essere caricato, può essere che un altro programma acceda alla porta seriale e la blocchi.

Illustrazione 46 Errore - Problemi di caricamento sulla scheda

Questo errore si nota con l'errore can't open device. Si raccomanda di chiudere tutti gli altri programmi.

Questi sonogli errori più comuni che si possono verificare trttando con Arduino. Naturalmente, non sono tutti. Se incontri un errore non elencato, potrai trovare assistenza sul forum di Arduino, all'indirizzo: https://forum.arduino.cc/, oppure potrai fare una semplice ricerca su internet.

In generale, il sito ufficiale di Arduino offre molto aiuto. Inoltre, si possono trovare molti progetti e istruzioni.

8. eBook gratuito

Grazie per aver comprato questo libro. Poiché il libro è stampato direttamente da Amazon e non ho alcuna influenza sulla qualità delle immagini, è possibile che alcuni dettagli vadano persi.

Ecco perché offro l'eBook a colori gratuitamente come file PDF quando si acquista il libro. Lì troverai tutte le immagini in alta risoluzione e avrai sempre l'ultima versione disponibile.

Per farlo, invia un messaggio con oggetto "Arduino-eBook" e uno screenshot dell'acquisto o una prova d'ordine al seguente indirizzo email (i dati privati possono essere redatti):

BenjaminSpahic@pbd-verlag.de

Ti invierò l'eBook immediatamente.

Se senti che manca qualcosa, il libro non è di tuo gradimento, hai dei suggerimenti per migliorarlo o domande da porre, sentiti libero di mandarmi una mail.

Il miglioramento dipende dalla critica costruttiva. Sto costantemente rivedendo il libro e sono felice di rispondere a qualsiasi suggerimento costruttivo di miglioramento.

Se il libro ti è piaciuto, invece, sarei felice di ricevere una recensione positiva su Amazon. Questo aiuta la visibilità del libro ed è la più grande lode che un autore possa ricevere.

ITuo Benjamin

Arduino senza conoscenze preliminari è la traduzione del bestseller tedesco "Arduino ohne Vorkenntnisse". Pertanto, alcuni passaggi del libro si concentrano sulla regione tedesca. Questo è completamente irrilevante per il contenuto, il modo di funzionamento e la comprensione generale.

Se avete trovato un errore linguistico, vi preghiamo di segnalarcelo tramite l'indirizzo e-mail indicato nell'impronta.

L'autore

Benjamin Spahic è nato a Heidelberg nel 1995 ed è cresciuto in un villaggio di 8.000 abitanti vicino a Karlsruhe. La sua passione per la tecnologia si riflette nei suoi studi come ingegnere elettrico con un focus sulla tecnologia dell'informazione presso l'Università di Scienze Applicate di Karlsruhe.

Ha poi approfondito le sue conoscenze in un programma di Master nel campo della produzione di energia rigenerativa presso l'Università di Scienze Applicate di Karlsruhe.

Com'è la vita quotidiana di un autore? Seguiteci ora su Instagram!

BENJAMIN.SPAHIC.AUTOR

Impronta:

PBD Verlag

Autore: Benjamin Spahic

Indirizzo:
Benjamin Spahic

Konradin-Kreutzer-Str. 12

76684 Östringen

Redazione/ correzione di bozze: Mentorium GmbH

Copertina: Kim Nusko

ISBN Paperback: 9798692381279

ISBN Hardcover:9798730263024

EMail: BenjaminSpahic@pbd-verlag.de

Facebook: Benjamin Spahic

Arduino senza conoscenze precedenti

Prima pubblicazione 02.10.2020

Distribuzione attraverso kindledirectpublishing

Amazon Media EU S.à r.l., 5 Rue Plaetis, L-2338, Lussemburgo

Disclaimer

L'autore non si assume alcuna responsabilità per l'attualità, la completezza e la correttezza delle informazioni fornite. Inoltre, nessuna garanzia può essere data per il raggiungimento delle competenze descritte.

Crediti fotografici:

Icone:
https://icons8.de/icon/113140/kugelbirne
https://icons8.de/icon/79638/obligatorische
https://icons8.de/icon/78038/math
https://icons8.de/icon/42314/taschenrechner

Tutti i contenuti non menzionati sono stati creati dall'autore stesso. È quindi l'autore della grafica e ha i diritti di utilizzo e distribuzione.

Illustrazione 1: https://commons.wikimedia.org/wiki/File:Z3_Deutsches_Museum.JPG*

Illustrazione 3: https:*
*//de.wikipedia.org/wiki/Datei:Widerst%C3%A4nde.JPGAbbildung 4**: https://de.wikipedia.org/wiki/Datei:Resistor_symbol_America.svg*

*Illustrazione 6**: https://de.wikipedia.org/wiki/Datei:Diodenalt2.png*

Illustrazione 8: https://de.wikipedia.org/wiki/Datei:Diode_pinout_de.svg*

Illustrazione 10: https://de.wikipedia.org/wiki/Datei:Elko-Al-Ta-Bauformen-Wiki-07-02-11.jpg*

Illustrazione 11: https://commons.wikimedia.org/wiki/File:Kondensatoren-Schaltzeichen-Reihe.svg*

*Illustrazione 15**:https://commons.wikimedia.org/wiki/File:AVR_group.jpg*

*Illustrazione 16**:https://commons.wikimedia.org/wiki/File:ArduinoUnoSMD.jpg*

*Illustrazione 17**: https://de.wikipedia.org/wiki/Datei:Arduino_uno_r3_isometr.jpg*

*Illustrazione 18**:https://commons.wikimedia.org/wiki/File:ArduinoUnoSMD.jpg*

*Illustrazione 19**: https://store.arduino.cc/arduino-uno-rev3*

** Questo file è reso disponibile sotto la GNU Free Documentation License.*

https://commons.wikimedia.org/wiki/Commons:GNU_Free_Documentation_License,_version_1.2Es può essere stato modificato.

*** Questo file è reso disponibile sotto la licenza Creative Commons "CC0 1.0 rinuncia al copyright".*

https://creativecommons.org/publicdomain/zero/1.0/deed.deEs modifiche possono essere state fatte.